Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

IVAN MATAK

RAZVOJ APLIKACIJE U OKRUŽENJU VUE.JS

Završni rad

Pula, rujan 2018.

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

IVAN MATAK

RAZVOJ APLIKACIJE U OKRUŽENJU VUE.JS

Završni rad

JMBAG: 0303061164, redoviti student

Studijski smjer: Sveučilišni preddiplomski studij Informatika

Kolegij: Programiranje

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijsko-komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: doc. dr. sc. Tihomir Orehovački

Pula, rujan 2018.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Ivan Matak, kandidat za prvostupnika informatike ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima, te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, rujan, 2018. godine

IZJAVA

o korištenju autorskog djela

Ja, Ivan Matak dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom „Razvoj aplikacije u okruženju Vue.js“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli,_____(datum)

Potpis

_____

Sadržaj

1. Uvod

Posljednjih godina postoji velika potreba za visoko sofisticiranim i kompleksnim web aplikacija koje bi trebale zamijeniti starije desktop aplikacije u svim područjima [2]. Internet je u zadnjih 10 godina postao glavni medij te je potpuno zasjenio starije oblike medija kao što su radio, novine i televizija. U poslovnome svijetu, svako ozbiljnije poduzeće danas ima vlastitu web stranicu ili aplikaciju kako bi potencijalni poslovni partneri ili kupci mogli pregledati i dobiti informacije o poduzeću.

Web trgovine su danas jako zastupljene te omogućuju jednostavnije postavljanje narudžbe nego telefonskim putem jer kupci mogu odmah pogledati koji su proizvodi dostupni te njihovu količinu, cijenu, dimenzije i slično. Drugi oblik poslovanja i dobivanja profita je razvoj aplikacija za računala ili mobilne uređaje. Način na koji se dobiva profit putem web stranica i aplikacija su reklame. Slično kao na televiziji, reklame su plaćene te se nalaze na sučelju aplikacije pa profit ovisi i o broju klikova na te reklame. Velika većina mobilnih aplikacija i igara imaju i opciju gdje se plaćanjem određenog iznosa reklame uklone sa korisničkog sučelja. Mobilne igre često imaju freemium način poslovanja. Dio igre je besplatan, a ukoliko se plati određeni iznos, moguće je odigrati cijelu igru uz neka poboljšanja ili bez reklama.

U zadnjih 10 godina web stranice i web aplikacije su postale mnogo dinamičnije i moćnije uz pomoć programskog jezika JavaScript. JavaScript je jedan od najpopularnijih programskih jezika na svijetu, te je specifičan po tome što se prikazuje u web pregledniku pošto je jedan od esencijalnih tehnologija World Wide Web-a uz HTML i CSS. Velika većina web preglednika podržavaju JavaScript koji se koristi za veliki udio web stranica. Velika količina programskog koda koja se inače nalazila na serverima je pomaknuta u internet preglednike što rezultira tome da aplikacije imaju tisuće i tisuće linija JavaScript koda i datoteka koje su povezane sa HTML i CSS datotekama bez prave i formalne organizacije. Upravo je to razlog zašto sve više i više developera koriste JavaScript framework za gradnju korisničkog sučelja i organizaciju datoteka i podataka. U web framework ubrajamo React.js, Angular i Vue.js. Svaki od ovih sučelja imaju svoje prednosti i mane te sličnosti i različitosti. Prije odluke o sučelju potrebno je napraviti analizu ponuđenih sučelja kako bi se utvrdilo koje sučelje najviše odgovara korisničkim potrebama.

Motivacija za implementaciju aplikacije StudentEvent potiče od želje da se na jednostavan i intuitivan način mogu kreirati događaji za studente koji tako mogu upoznati ljude sa sličnim interesima. Na taj način se olakšava način organiziranja događaja za studente pošto se još uvijek veliki broj događaja promovira pomoću plakata. Za razvoj aplikacije se koristi sučelje Vue.js koje je odabrano nakon analize i usporedbe sličnih sučelja za razvoj aplikacija, a ta analiza je opisana u uvodnim poglavljima. Uvodna poglavlja se odnose i na istraživanje tržišta gdje se prikazuju i kritički analiziraju slične aplikacije.

Zatim se kreće na implementaciju aplikacije StudentEvent gdje će se u trećem poglavlju prikazivati postupak implementacije osnovnog oblika aplikacije koji se odnosi na alatnu traku i naslovnu stranicu. U trećem se poglavlju se provodi i implementacija routing-a za povezivanje stranica. Četvrto i peto poglavlje se fokusiraju na implementaciju izgleda ostalih stranica te implementaciju mogućnost lokalnog spremanja podataka pomoću Vuex-a. Peto poglavlje prikazuje postupak implementacije forme za nove događaje, a u šestom poglavlju se predstavlja baza podataka i postupak implementacije stranica za prijavu, odjavu i registraciju za koje je i potrebna implementirana baza podataka. Sedmo poglavlje prikazuje na koji način se događaji mogu spremati na bazu podataka te kako se pomoću te iste baze može promijeniti lozinka korisnika. Osmo poglavlje prati implementaciju različitih mogućnosti za događaje kao što su uređivanje naslova, opisa, kapaciteta, vremena i datuma, brisanje događaja. Zadnje poglavlje se odnosi na dovršavanje aplikacije sa implementacijom mogućnosti prijave i odjave na događaj te dohvaćanje trenutno prijavljenih događaja za trenutno prijavljenog korisnika.

2. Usporedba sučelja i analiza tržišta

U ovom poglavlju će se uspoređivati različita sučelja za razvoj aplikacija te će se provesti analiza tržišta. Analiza tržišta je potrebna kako bi sa ekonomskog aspekta prepoznali prilike i prijetnje tržišta.

2.1. Usporedba sučelja

Angular ima jako dobar koncept za korištenje, a u to spadaju modularnost te način na koji se komponente koriste sa modulima. Isto tako, Angular omogućuje visoku razinu kontrole, te se lako mogu dodavati routing i form validacije. Angular 1 se još uvijek koristi na mnogo web stranica, a Angular 2 je u teoriji brži od svog prethodnika, no imao je problema tijekom razvoja, iako je vlasnik Angular sučelja Google. Naime, Angular je podložan naglim promjenama tijekom svojeg razvoja koje se događaju i danas. Nagle promjene u sučeljima su loše zato što zahtijevaju od korisnika konstantno ponovno učenje koje je u krajnju ruku nepotrebno pošto bi se takva sučelja trebala postupno nadograđivati bez naglih i jako drastičnih promjena. Glavni je problem ako razvijate web stranicu, ona nakon jedne godine neće biti funkcionalna do neke razine upravo zbog naglih promjena koje Angular izbacuje ili mijenja u svojim nadogradnjama. Zbog toga development tim za Angular radi na kompatibilnosti sa starijim verzijama sučelja i na stabilnosti iste. Angular isto tako koristi TypeScript programski jezik koji je po sintaksi sličan programskim jezicima kao što je C#.

React.js je poznat po tome što ga koristi društvena mreža Facebook. React.js koristi kombinaciju JavaScript i HTML jezika. No problem je što React.js koristi JSX koji predstavlja ekstenziju JavaScript jezika. Problem je što JSX ograničava pune mogućnosti HTML i CSS jezika, odnosno mijenja se njihova sintaksa dok JavaScript ostaje nepromijenjen. Upravo je zato React.js malo teže sučelje za naučiti, pogotovo za početnike. Usprkos tome, React.js je odlično sučelje, no zahtijeva da se zna JSX. React.js ima puno manje potrebe za optimizacije za web stranice zato što je sve već u JavaScript jeziku za razliku od sučelja Angular. No, Angular ima odvojene HTML i TypeScript dijelove koda što znači da se može iskoristiti puni potencijal HTML jezika.

Između ova dva sučelja, sve ovisi o osobnoj preferenciji te koji stil programiranja više odgovara pojedincu.

Vue.js predstavlja kombinaciju dijelova iz sučelja Angular i React.js. Razlog je zato što se može koristiti više jezika kao što su ES5, ES6 i TypeScript. No za razliku od sučelja Angular i React.js, Vue.js je vrlo fleksibilno i lagano sučelje za brzu implementaciju prototipova korisničkih sučelja. U procesu implementacije prototipova isto tako nudi lagani i fleksibilni način za povezivanje podataka i ponovno korištenje postojećih komponenti [1]. Ima i odvojeni JavaScript, HTML i CSS kod što je dobro jer pruža bolju preglednost cjelokupnih datoteka i koda. Vue.js ima najdetaljniju dokumentaciju od svih navedenih sučelja te službene pakete za routing i state management, kao i Angular. No, iza sučelja Vue.js ne stoji velika kompanija kao što su kod sučelja Angular i React.js. Prema podacima o broju preuzimanja sa stranice GitHub, React.js je najpopularniji JavaScript front end Framework, dok je Vue.js jedan od najbrže rastućih sučelja prema popularnosti na toj stranici. Angular čiji je vlasnik Google i koji koristi Microsoft-ov jezik TypeScript je najstabilnije sučelje [2].

Usprkos tome, za Vue.js postoji paket za materijalni dizajn komponenti koji se naziva Vuetify. U dokumentaciji za Vuetify se nalazi popis komponenti te primjeri programskog koda koji prikazuju kako se te komponente i koriste. Ovaj paket je presudni faktor odluke za razvoj aplikacije u sučelju Vue.js zato što omogućuje brzu i jednostavnu izgradnju dizajna stranica pomoću već programiranih komponenti koje se mogu prilagođavati prema vlastitim potrebama. Vuetify se isto tako redovito ažurira te se poboljšavaju i dodaju nove komponente. Upotreba sučelja Vue.js se proširila na sve veći broj kompanija, od kojih su najpopularnije Adobe i IBM [5]. Razlog za odabir Vue.js sučelja je i odvojenost JavaScript koda na način da će se dijelovi koda pozivati iz js datoteka kako bi se koristile za Vue datoteke na kojima se nalaze komponente i podaci.

2.2. Istraživanje tržišta

Aplikacija koja se implementira i analizira se naziva StudentEvent. Aplikacija StudentEvent je namijenjena svim studentima kako bi se organizirala razna druženja i upoznavanja novih ljudi sa sličnim interesima. Pogotovo je korisna jer omogućuje novim studentima lakšu prilagodbu na nova okruženja pošto promjenom sredine se teško nalazi novo društvo.

Istraživanjem tržišta moguće je utvrditi postoje li već slične stranice ili aplikacije koje imaju slične funkcionalnosti. Web stranica www.eventbrite.com nudi sličnu funkcionalnost u smislu da se mogu pretraživati i vidjeti razne informacije o događajima. No, ova stranica nije besplatna za korištenje ukoliko želite organizirati vlastite događaje. Postoje razni planovi koji se plaćaju na mjesečnoj razini kako bi se mogle koristiti neke osnovne funkcionalnosti kao što je kreiranje događaja.

Slične funkcionalnosti ima društvena mreža Facebook koja omogućuje kreiranje događaja i unos detalja o događajima. Događaji se mogu dijeliti, no nisu potpuno javni zbog toga što se mogu samo pozvati prijatelji na toj društvenoj mreži. Ukoliko se događaj ne bude dalje dijelio, njegov doseg neće biti vrlo velik.

Istraživanje pokazuje kako već postoje stranice i aplikacije koje omogućuju kreiranje događaja, no ne postoji besplatna aplikacija te kategorije koja je usmjerena isključivo prema studentima.

3. Implementacija aplikacije StudentEvent – osnovni oblik aplikacije

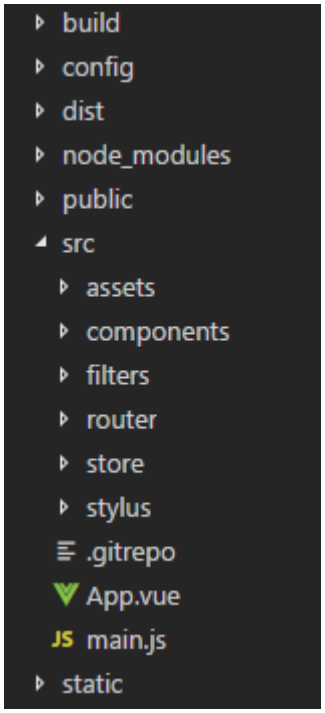
Aplikacija StudentEvent će koristiti okruženje Vue.js, paket komponenti Vuetify i Google Firebase koji će poslužiti kao baza podataka. Komponente predstavljaju dijelove aplikacije koje sadrže vlastite podatke te se mogu ponovno koristiti u različitim dijelovima aplikacije [1]. Mogućnosti aplikacije su pregled događaja, prijava na događaj, odjava sa događaja, brisanje događaja, registracija, uređivanje podataka za prijavu te postavljanje događaja u kojem se mogu birati i uređivati vrijeme, lokacija, kapacitet, opis te prenijeti slika događaja.

3.1. Struktura i gradnja osnovnog oblika aplikacije

Kako bi se započelo sa implementacijom, potrebno je instalirati okruženje Vue.js. Upute za instalaciju Vue.js okruženja se mogu naći na njihovoj službenoj stranici [19]. Upute za instalaciju paketa komponenti Vuetify se isto tako mogu naći na službenoj stranici [15]. Nakon instalacije potrebnih komponenti, otvara se mapa u kojoj se nalazi projekt u programu za uređivanje po vlastitom izboru, što bi u ovom slučaju bio Visual Studio Code. JavaScript programski jezik će se koristiti u svrhu pisanja funkcija pomoću kojih će se izvršavati sve ključne radnje u aplikaciji, a Vuetify će se koristiti unutar Vue datoteka za izgradnju i slaganje komponenti aplikacije. Unutar mape aplikacije nalazi se struktura aplikacije.

Mape „build“, „conifg“ i „node_modules“ služe za pohranjivanje podataka o učitavanju aplikacije, odnosno u tim mapama se nalaze instalirano okruženje Vue.js i paket komponenti Vuefity. U mapama „dist“ i „public“ se nalazi verzija aplikacije koja je spremna za deployment na internet, no o tome kako su dobivene te mape će se razjasniti kasnije. Mapa „src“ sadržava razne druge mape koje zapravo dijele aplikaciju kako bi se lakše organizirale datoteke. Podmapa „assets“ služi za postavljanje raznih slika ili nekih drugih podataka za aplikaciju, dok je podmapa „components“ najraširenija po svojoj upotrebi jer unutar nje se dijeli aplikacija prema osobnim preferencijama na stranice i dijaloge koji se koriste. Podmapa „filters“ služi za transformaciju formata podataka, to će se kasnije koristiti kod standardizacije i lokalizacije formata datuma. Jako bitne mape „router“ i „store“ se koriste na način da

„router“ preusmjerava na razne stranice koje su definirane u mapi „components“, dok „store“ služi za pohranjivanje svih JavaScript funkcija što predstavlja samu srž svih funkcionalnosti aplikacije. „Stlyus“ mapa uglavnom sadržava boje koje se koriste u aplikaciji. Na kraju, u mapi „src“ se još nalaze datoteke „App.vue“ i „main.js“. Datoteka „App.vue“ sadrži osnovnu strukturu aplikacije što se tiče dizajna, a „main.js“



Slika 1. Struktura glavnih mapa aplikacije StudentEvent

zapravo pokreće sve komponente prilikom pokretanja aplikacije, odnosno u toj datoteci se definiraju sve datoteke s kojima će se raditi unutar aplikacije. Mapa „static“ služi uglavnom za neke statične podatke koji su potrebni za pokretanje aplikacije.

Nakon specificiranja mapa i njihovih uloga u izgradnji aplikacije, uređuje se App.vue datoteka koja se nalazi u „src“ mapi. App.vue ima svoj zadani Vuetify setup, no taj zadani dio se briše kako bi mogli početi programirati sa čistom i praznom stranicom.

Property <template> označava granicu i prostor u kojem se nalazi kod i komponente za aplikaciju, dok property <script> označava prostor za dodjeljivanje vrijednosti

App.vue

```
<template>
  <v-app>

  </v-app>
</template>

<script>
  export default {
    data () {
      return {

      }
    }
  }
</script>

<style lang="stylus">
  @import './stylus/main'
</style>
```

Slika 2. Osnovni oblik datoteke App.vue za praznu stranicu

raznim varijablama ili za pozivanje funkcija koje se nalaze u JavaScript datotekama. Za kreiranje dobrog korisničkog iskustva, bitno je odabrati odgovarajuću arhitekturu aplikacije [2]. Pošto se radi glavno korisničko sučelje za aplikaciju u single page obliku, postavlja se i Vuetify komponenta <v-app> koja će se odnositi na sve ostale povezane Vue stranice. Property <style> uvozi definirane stilove iz druge datoteke, a uglavnom se koristi za boje.

Prvo se postavlja HTML element <main> u kojem će se nalaziti glavni sadržaj aplikacije, dok će van tog elementa biti izbornik, odnosno toolbar koji je statični dio, a ne dinamički kao ostali sadržaj stranice. Za implementaciju toolbar izbornika, prvo se postavlja <v-toolbar> koji je Vuetify komponenta. Unutar <v-toolbar> komponente se dodaje <v-toolbar-title> komponenta [17]. Unutar <v-toolbar-title> elementa jednostavno se postavi naslov aplikacije, što je u ovom slučaju „StudentEvent“. Kako bi se dodali ostali dijelovi toolbar komponenti, jednostavno se koriste <v-btn> komponente, koje predstavljaju gumbe [7]. No, prije nego što se postave gumbi,

okružiti će se sa komponentom <v-toolbar-items> koji služi da se postave sve ostale komponente unutar jednog izbornika ili toolbar-a. Dodaje se broj komponenti gumba koji je i potreban za aplikaciju, a novi gumbi se mogu dodati kasnije. Kako bi gumbi izgledali bolje, unutar komponente <v-btn> upisuje se „flat“ što zapravo miče border koji gumbi inače imaju. Gumbi će u ovom slučaju biti odmah nakon naslova aplikacije, a kako bi to pomaknuli, između <v-toolbar-title> i <v-toolbar-item> komponenti se postavlja komponenta <v-spacer> koja stvara prostor između gumbi i naslova. Nakon što se poredao naslov i gumbi, dodaju se ikone za gumbe i to na način da se dodaje komponenta <v-icon> unutar <v-btn> komponente. Popis ikona, njihov naziv i izgled je dostupan na službenoj dokumentaciji [3]. Ime ikone se postavlja unutar <v-icon> komponente. Radi ljepšeg izgleda, unutar <v-icon> komponente se postavlja i riječ „left“ kako bi se ikona pomakla ulijevo tako da nije direktno povezana sa tekstom.

Nakon što se postavio osnovni izgled toolbar-a, red je da se postavi i ikona sa lijeve strane kako bi mogli koristiti aplikaciju na mobilnim uređajima i dinamički prilagođavati aplikaciju na različite rezolucije ekrana. Za početak, prije <v-toolbar> komponente, dodaje se <v-navigation-drawer> komponenta koja treba biti povezana se nečim što ju poziva. Vue.js omogućuje povezivanje data modela sa prezentacijskim slojem aplikacije. Isto tako omogućava ponovnu upotrebu komponenti kroz proces implementacije aplikacije [1]. U tu svrhu se unutar te komponente postavlja „v-model“ koji služi za povezivanje sa nekom drugom komponentom koja je tipa boolean kako bi se znalo kada će se ta komponenta prikazati. Za naziv „v-model“ se postavlja varijabla „sideNav“ te će se ista definirati unutar skripte pod data return, zato što je sideNav varijabla, te ju je potrebno definirati i dodijeliti joj vrijednost koja će biti „false“. Sada je potrebna neka komponenta koja će tu varijablu pozvati te tako promijeniti joj vrijednost kako bi se <v-navigation-drawer> komponenta mogla prikazati. U tu svrhu se dodaje <v-toolbar-side-icon> komponenta iznad <v-toolbar-title> komponente kako bi prikazali ikonu sa tri crtice. Dodaje se i „@click.stop“ listener koji se okida kada korisnik klikne tu ikonu te se taj izbor zadržava, što je razlog dodavanja riječi „stop“. Vrijednost listener-a se postavlja na „sideNav = !sideNav“ što znači da dodaje varijabli sideNav suprotnu bool vrijednost s obzirom na trenutnu vrijednost. Sada bi se trebala dodavati lista za taj izbornik sa strane. To se izrađuje tako što unutar <v-navigation-drawer> komponente

se postavi <v-list> komponenta, te unutar nje <v-list-tile> komponenta koja predstavlja jedan redak u listi [12]. Unutar te komponente je potrebna <v-list-tile-action> komponenta koja sadržava komponentu ikone, a komponenta <v-list-tile-content> sadržava naziv tog retka. Sada kada su izrađeni drawer i toolbar izbornika, potrebno je postaviti kako će se taj toolbar prikazivati ovisno o rezoluciji. Kako bi se

```
<template>
  <v-app>
    <v-navigation-drawer v-model="sideNav">
      <v-list>
        <v-list-tile>
          <v-list-tile-action>
            <v-icon>supervisor_account</v-icon>
          </v-list-tile-action>
          <v-list-tile-content>Test</v-list-tile-content>
        </v-list-tile>
      </v-list>
    </v-navigation-drawer>
    <v-toolbar dark class="primary">
      <v-toolbar-side-icon
        @click.stop="sideNav = !sideNav"
        class="hidden-sm-and-up "></v-toolbar-side-icon>
      <v-toolbar-title>StudentEvent</v-toolbar-title>
      <v-spacer></v-spacer>
      <v-toolbar-items class="hidden-sm-and-down">
        <v-btn flat>
          <v-icon left dark>supervisor_account</v-icon>
          Test
        </v-btn>
      </v-toolbar-items>
    </v-toolbar>
    <main>

    </main>
  </v-app>
</template>

<script>
  export default {
    data () {
      return {
        sideNav: false
      }
    }
  }
</script>
```

Slika 3. Izgled koda za datoteku App.vue

to postiglo, jednostavno se dodaje CSS klasa „hidden-sm-and-up“ kako bi ostala skrivena kod malih i većih uređaja. A unutar <v-toolbar-items> postavlja se CSS klasa „hidden-sm-and-down“ kako bi se ikone i gumbi u glavnom toolbar izborniku sakrile na manjim uređajima. Kako bi se lakše upravljalo bojama, u datoteku „main.styl“ je potrebno dodati podatke iz Vuetify dokumentacije koji daju strukturu i kod za uvoz boja [16]. Naravno, mogu se birati sve boje koje se isto tako nalaze u dokumentaciji. Sada, ako se unutar <v-toolbar> komponente doda class = „primary“, boja će biti ona koja je zadana u datoteci za stil pod nazivom „primary“. Kako bi slova bila prave boje s obzirom na kontrast, može se dodati i riječ „dark“ što znači da je pozadinska boja tamna, pa će slova biti bijele boje radi bolje vidljivosti.

3.2. Učitavanje izbornika i routing stranica

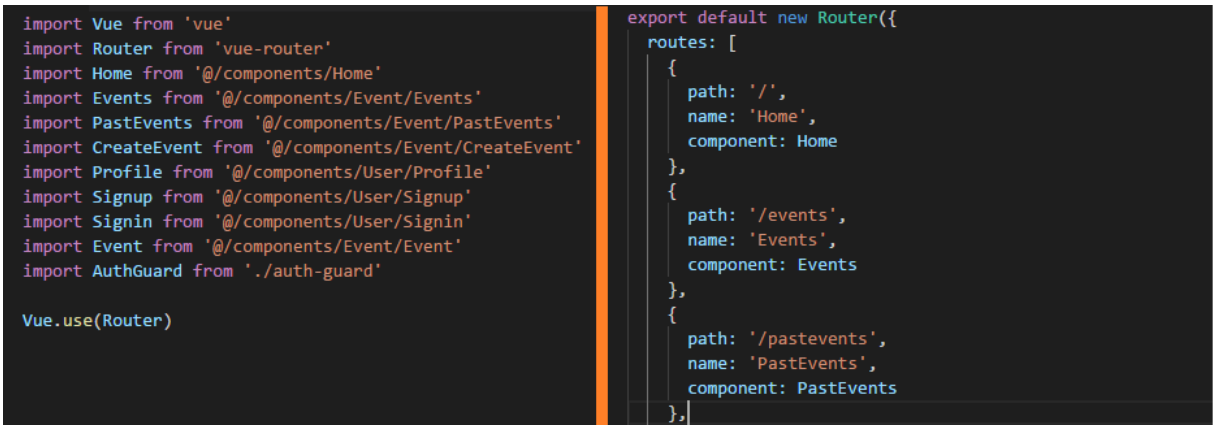
Za učitavanje izbornika potrebno je polje, pa će se unutar data return djela napraviti polje pod nazivom menuItems. Razlog tome je da se ne moraju pisati toolbar itemi te da je moguće upravljati podacima na izborniku s obzirom na to je li korisnik ulogiran ili nije. Unutar tog novog polja, postavlja se JavaScript objekt koji će sadržavati ikonu i naslov, odnosno title. I tako se izrađuju objekti za svaki gumb koji je poželjno imati u izborniku, kao što su gumbi za prijavu i registraciju. Nakon što se polje popunilo, potrebno ga je dinamički ispisati, a to se radi pomoću „v-for“ petlje koja se postavlja unutar <v-list-tile> komponente. Kako bi se prolazilo dinamički kroz polje, potrebno je postaviti naredbu „v-for“ na slijedeći način: v-for = “item in menuItems“ :key=“item.title“. Na ovaj način se određuje binding za unikatni key koji je u ovom slučaju naslov pošto je svaki naslov različit. Sada se mogu zamijeniti nazivi za gumbe koji su pisani manualno sa {{ item.title }} unutar <v-list-tile-content> te {{ item.icon }} unutar <v-icon>, i to se sve izvlači dinamički iz polja koje je napravljeno ranije. To je bilo raspoređivanje podataka u izborniku sa strane, a za toolbar izbornik potrebno je na sličan način postaviti „v-for“ za „v-btn“ unutar „<v-toolbar-items>“ komponente. Pa onda se mijenjaju statični podaci na isti način unutar <v-btn> i <v-icon> komponenti.

Sada kada postoji uređeni dizajn oba izbornika, potrebno je te gumbe preusmjeravati na njihove odgovarajuće stranice. Mapa „router“ već postoji ako je tako odlučeno pri instalaciji tako da već postoji osnova za preusmjeravanje. U datoteci App.vue će se unutar <main> komponente dodati <router-view> komponenta kako bi se mogao povezati router. Sada je potrebno implementirati glavne stranice koje će se koristiti u aplikaciji. Pametno bi bilo unutar mape „components“ napraviti podmape na način da se zna za što se koriste. Za početak, podmape će se zvati „Event“ i „User“ tako da postoje stranice koje se odnose na korisnika i na neki događaj te će se tako i raspodijeliti. Zatim je potrebno kreirati .vue datoteke za svaku stranicu, te u svakoj stranici se stavlja običan HTML <p> paragraf kako bi se znalo koja se stranica prikazuje. Naravno, stranice se naknadno mogu dodavati po želji i potrebi, sada je vrijeme da se napravljene stranice usmjere.

Kako bi se razumjelo kako to funkcionira, prelazi se u „main.js“ datoteku gdje se vidi kako je router uvezen u projekt na samom vrhu datoteke. Router je isto tako

dodan u novu Vue instancu u toj datoteci pa je povezan sa cjelokupnim projektom. Sada se u index.js router datoteci dodaju nove rute, odnosno povezuju se stranice. Zadana ruta ima path na „/“ što je i zadana stranica projekta. No, komponenta će se zvati „Home“ kako bi se znalo da je to prva stranica koja se otvara kada se aplikacija pokrene. Na vrhu te datoteke potrebno je sve te komponente unijeti na način da se na vrhu napiše „import Home from '@/components/Home'“. Znak @ se referencira na mapu „src“ pa je potrebno definirati točan put do te datoteke. I na taj način se radi uvoz sve ostale stranice koje su dodane do sada. Ukoliko je to na primjer stranica za prijavu koja se naziva „Signup“, njezina će ruta za uvoz biti „import Signup from '@/components/User/Signup'“ jer se nalazi u mapi „User“. Nakon što je napravljen uvoz za sve stranice, dodaje se novi path za svaku od njih. Path se postavlja prema potrebama, no „name“ i „component“ moraju imati isto ime kao i uvezena stranica. Na kraju kada su dodane sve rute, postavlja se „mode: 'history'“ što miče znak „#“ iz url adrese.

Povezane stranice je potrebno i navesti u App.vue datoteci u menuItems polju. Dodaje se link property te se svaka stranica usmjerava na path koji je definiran u router-u. Isto tako se postavlja da naslov „StudentEvent“ klikom preusmjerava na početnu stanicu. Jednostavno na mjestu gdje je taj naslov se postavlja komponenta <router-link> i unutar te komponente se dodaje „to='/' “ koji preusmjerava na početnu stranicu te „tag='span' “ i „style='cursor: pointer' “ kako bi se promijenio oblik miša iznad naslova. Na kraju ih je još potrebno povezati sa komponentama u App.vue datoteci. Kod „v-for“ za oba izbornika se dodaje „router :to=item.link“ kako bi se povezali sa poljem. Na taj način se ostvaruju funkcionalne i povezane stranice sa izbornikom.



```
import Vue from 'vue'
import Router from 'vue-router'
import Home from '@/components/Home'
import Events from '@/components/Event/Events'
import PastEvents from '@/components/Event/PastEvents'
import CreateEvent from '@/components/Event/CreateEvent'
import Profile from '@/components/User/Profile'
import Signup from '@/components/User/Signup'
import Signin from '@/components/User/Signin'
import Event from '@/components/Event/Event'
import AuthGuard from './auth-guard'

Vue.use(Router)
```

```
export default new Router({
  routes: [
    {
      path: '/',
      name: 'Home',
      component: Home
    },
    {
      path: '/events',
      name: 'Events',
      component: Events
    },
    {
      path: '/pastevents',
      name: 'PastEvents',
      component: PastEvents
    },
```

Slika 4. Kod za datoteku "index.js" u mapi „router"

3.3. Uređivanje naslovne stranice

Sada kada su dovršene povezane stranice i osnovni izgled izbornika, vrijeme je da se uredi i dizajnira naslovna stranica. Naslovna stranica je zamišljena sa dva gumba koji vode na stranice za nadolazeće događaje i za kreiranje novog događaja te sa pokretnim slikama od događaja. Za dizajn stranice su jako korisne grid komponente koje dijele sadržaj stranice za više dijelova [11].

U datoteci Home.vue će se izgraditi izgled naslovne stranice. Zasada postoji samo <template> komponenta pa će se unutar nje dodati <v-container> koji okružuje podijeljeni grid za stranicu. Unutar te komponente se stavlja <v-layout row wrap class="mt-2"> što predstavlja jedan redak za podjelu sadržaja stranice te sadržaj će prijeći u novi red ako isti ne stane te klasu za prostor između komponenti. Nakon toga se postavlja <v-flex xs12 sm6 class="text-xs-center text sm-right"> komponenta unutar koje je moguće redati sadržaj i poravnati ga sa Vuetify klasom koja to omogućuje. Na ovaj način, prvi gumb će biti pomaknut na desni kraj svojeg prostora na većim uređajima, te će biti centriran na manjim uređajima. Za drugi gumb se postavlja riječ „left" zato što se nalazi u drugome flex prostoru te će na taj način biti dobro centrirani. Oznake „xs12" i „sm6" označavaju kako će se sadržaj ponašati s obzirom na veličinu uređaja. Na jako malim uređajima će zauzimati cijeli red, a na malo većim uređajima pola reda. Prvo se postavlja jedan gumb za pregled događaja pomoću komponente <v-btn large router to="/events" class="info"> jer cilj je da gumb bude velik da usmjerava na stranicu sa popisom događaja u boji klase info za stil. Za drugi gumb koji usmjerava na stranicu događaja, izrađuje se nova <v-flex xs12 sm6> komponenta sa istim postupkom, samo se usmjerava na drugu stranicu, u ovom slučaju „/event/new".

Nakon dva gumba, cilj je dodati carousel komponentu koja se isto tako nalazi u službenoj dokumentaciji [9]. Isto kao i ranije, postavljaju se <v-layout> i <v-flex xs12> komponente te <v-carousel> komponenta koja je kopirana iz dokumentacije. Nakon toga, potrebni su neki podaci kako bi se mogla pozvati petlja unutar carousel-a kako bi prikazali podatke. Za tu potrebu se na dnu datoteke otvara <script> i isto kao i prije dodaje se data return() dio koji sadrži varijable. Ponovno je potrebno polje koje će se zvati „events" te će se u to polje staviti imageUrl, title i id koji će biti izmišljeni, jer je cilj samo testirati kako carousel radi pošto još ne postoji baza za

spremanje tih podataka. Nakon što je polje popunjeno podacima, potrebno je urediti carousel.

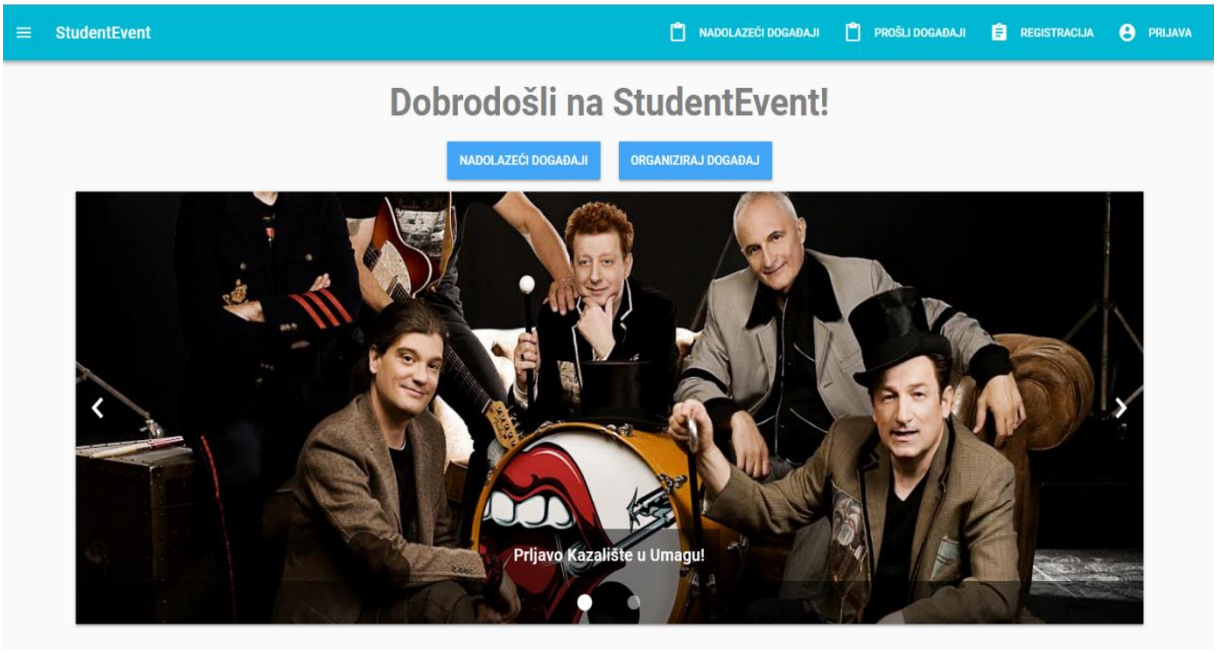
```
<v-carousel>
  <v-carousel-item
    v-for="(item,i) in items"
    :key="i"
    :src="item.src"
  ></v-carousel-item>
</v-carousel>
```

Slika 5. Trenutni izgled carousel-a iz dokumentacije

Prvo se mijenja „items“ u „events“, a „item“ se mijenja u „event“ te se miču zagrade i varijabla „i“. Postavlja se „:key = event.id“ te „:src = event.imageUrl“. Sada kada postoje slike čiji su linkovi kopirani sa interneta, postavlja se a unutar tog div-a „{{ event.title }}“ kako bi se prikazao i naslov događaja. Na dnu stranice se dodaje <style scoped> tako da se odnosi samo na tu stranicu, a unutar te komponente „.tite“ komponenta sa sljedećim opisom:

position: absolute;
bottom : 50px;
backgrond-color: rgba(0,0,0,0.5);
color: white;
font-size: 2em;
padding: 20px;

Na taj način je uređeno pozicioniranje za naslov od događaja, njegov padding, boju, veličinu fonta i slično. Još je potrebno postaviti naslovnu poruku, a za to jednostavno se kopira jedan postojeći <v-layout> te unutar njega se postavlja obični paragraf <p> sa porukom koja će se prikazati. Nakon ovoga napravljena je funkcionalna početna stranica i izbornici.



Slika 6. Izgled naslovne stranice

4. Implementacija aplikacije StudentEvent – popis događaja, zasebni događaj, Vuex i prošli događaji

U ovom poglavlju se opisuje postupak implementacije stranica koje prikazuju popis događaja, zasebni događaj, prošli događaji te Vuex kojim se upravlja stanjem podataka i biblioteka.

4.1. Uređivanje stranica za popis događaja i zasebni događaj

Prvo će se implementirati stranica za popis događaja za koju je već izrađena datoteka. Za raspored se koristi grid pa će se tako samo redati komponente <v-container>,<v-layout row wrap> i <v-flex xs12 sm10 md8 offset-sm1 offset-md-2>. Offset zapravo označava koliko će praznog prostora biti između rubova kartice. Događaji će se poredati u <v-card class="info"> komponente unutar <v-flex xs12 sm4 md3> [8]. Unutar card komponente se postavlja <v-container fluid> kako bi se iskoristio sav dostupni prostor. Zatim <v-layout row> i <v-flex xs5> pa unutar njih <v-card-media> koji ima svoje „src" i „height" property-e za uređivanje. Za „src" se koristi link slike koji se koristi i za carousel, a „height" se prilagođava prema potrebama korisnika. Ispod <v-flex> za card media, dodaje se još jedan <v-flex> za informacije o događaju. Za tekst iz ranije navedene dokumentacije može se koristiti <v-card-title primary-title>. Unutar te komponente postoji jedan koji sadrži <h5 class „white--text mb-0"> za naslov sa bijelim slovima te ispod njega još jedan za opis događaja, odnosno datum. Nakon informacija, potrebne su akcije za svaki događaj. Ispod card title komponente postavlja se <v-card-actions>, a unutar te komponente postavljaju se gumbi sa ikonicama <v-btn flat> sa tekstom te <v-icon left light> strelicu za ljepši izgled te da je sa lijeve strane i svijetlije boje prelaskom miša preko gumba.

Sada nam treba funkcionalnost gumba koji je prethodno dodan. Gumb treba voditi korisnika na taj specifični događaj. Prvo unutar router-a je potrebno dodati novu rutu čiji je path „/events/:id", a name i component su oba „Event". Na početku te datoteke isto tako je potrebno napraviti uvoz (import) za tu stranicu te ju kreirati ukoliko ona ne postoji. U novoj datoteci se redom kao i prije postavljaju <template>, <v-container>, <v-layout row wrap> i <v-flex x12>. Unutar flex komponente se ponovno koriste <v-card> i <v-card-title> u kojima će se postaviti <h6 class="primary-

-text“> za naslov događaja. Ispod <v-card-title> komponente postavlja se <v-card-media> čiji će se sadržaj kopirati iz datoteke Events.vue te height property se postavlja na 400px. Ispod te komponente slijedi <v-card-text> koji sadrži gdje se postavlja neki tekst za test. Nakon toga se stavlja <v-card-actions> i unutar toga <v-spacer> i <v-btn class=“primary“> za registraciju za događaj. Sada se preusmjerava klik sa popisa događaja kako bi se usmjerio kako treba, zasada statično, a kasnije dinamički preko baze podataka. U datoteci Event.vue u onom gumbu za pregled jednog događaja stavlja se „to='events/1'. Nakon toga se prelazi u Home.vue gdje već postoji statički id za testni događaj, pa se taj id kopira i postavlja pod <v-carousel-item> „@click='onLoadEvent(event.id)'“. To je funkcija pa ju je potrebno pozvati i definirati na dnu datoteke po „methods“. Funkcija glasi ovako:

```
onLoadEvent (id) {
    this.$router.push('/events' + id)
}
```

Za carousel se još može dodati i style=“cursor: pointer;“ zato što sada se može kliknuti i usmjeriti se na događaj preko carousel-a bazirano na id-u događaja koji je zasada statičan.

4.2. Vuex state management

Kako bi se koristile funkcije i iste spremale u zasebne datoteke za uredniji kod, mora se koristiti Vuex state management [18]. Instalacija je vrlo jednostavna te se samo koristi jedna naredba: „npm install –save vuex“. Sada je potrebno kreirati novu mapu unutar „src“ mape pod nazivom „store“ te unutar te mape datoteku „index.js“. Unutar te datoteke, radi se uvoz za Vue i Vuex. Vuex se tada poziva sa „Vue.use(Vuex)“ te se onda može početi koristiti. Prvo se treba postaviti neka konstanta koja će biti store za sve funkcije pomoću Vuex-a. To se radi sa slijedećom naredbom: „export const store = new Vuex.Store()“. Store će unutar sebe sadržavati state, mutations, actions i getters. State nam govori o stanju podataka, mutations služi za promjenu tih podataka, actions služe kako bi se mutations izvršile, a getters služe kako bi pozvali store unutar komponenti. Unutar state-a se deklarira polje pod

nazivom „loadedEvents“ u kojem se postavljaju i kopiraju sve informacije o statičnim događajima iz Home.vue datoteke kako bi preko tog store-a bio omogućen pristup tim podacima iz bilo koje datoteke. Zatim van tog polja se postavlja i korisnik (user) koji će imati svoj statički id koji se postavi prema vlastitom izboru, te polje „registeredEvents“ koje označava na koje je događaje korisnik prijavljen. Kako bi se ti podaci mogli koristiti, potrebno je postaviti getter-e. Getter se mora zvati kao i mutation, a to je „LoadedEvents“. Kod mora dohvatiti i sortirati događaje po datumu, u tu svrhu unutar događaja se dodaje property za datum ukoliko to još nije učinjeno ranije.

Funkcija vraća state u kojem su događaji, te ih uspoređuje i sortira. Zatim se radi još jedan getter koji dohvaća id pojedinog događaja. Za njega se koristi Javascript funkcija find.

```
getters: {
  loadedEvents (state) {
    return state.loadedEvents.sort((eventA, eventB) => {
      return eventA.date > eventB.date
    })
  },
  loadedEvent (state) {
    return (eventId) => {
      return state.loadedEvents.find((event) => {
        return event.id === eventId
      })
    }
  }
}
```

Slika 7. Getter-i za događaje

Sada bi se ti getter-i trebali i koristiti, tako da se prelazi u datoteku Home.vue te na dnu datoteke se briše „data“ dio koda, te umjesto „data“ se postavlja „computed“ kako bi se podaci dohvatili. Unutar computed funkcije „events“ potrebno je pozvati taj getter na sljedeći način: „return this.$store.getters.featuredEvents“. Taj getter se izrađuje tako da se u return postavi getters.loadedEvents.slice(0,5). Tako se prilagodio prikaz za carousel da se ne prikazuju baš svi događaji. Kako bi sve to radilo, potrebno je u main.js datoteci napraviti uvoz na vrhu datoteke „import {store} from './store'“ te jednostavno dodati store ispod router-a kako bi postao dio cjelokupnog projekta.

Učitavanje događaja se radi preko carousel-a na naslovnoj stranici, sada bi bilo potrebno isto napraviti i za stranicu sa popisom događaja, odnosno datoteku „Events.vue“. Na dnu datoteke radi se <script> te opet computed sa funkcijom „events“ koja vraća iz getter-a sve događaje, a ne kao prijašnji primjer gdje je broj

događaja ograničen. U istoj datoteci pri vrhu pod prvi <v-layout> dodaje se v-for=“event in events“ :key=“event.id“ class=“nb-2“. Nakon toga redom kao i prije mijenja se naslov sa {{ event.title }}, izvor slike sa :src=“{{ event.imageUrl }}“, te datum sa {{ event.date }}. Još nedostaje mogućnost da klik na pregled događaja vodi točno na taj specifični događaj. Tako da se odlazi pod taj gumb i unosi sljedeći kod: <v-btn flat :to=“ '/events/' + event.id“>. Nakon ove radnje više nije poželjno učitavati statičke događaje, već je to poželjno napraviti pomoću store-a. Na dnu datoteke Event.vue se postavlja <script> te pod computed se radi funkcija „event“ kojom se poziva funkcija iz store-a „loadedEvent“ kojoj se unutar zagrada prosljeđuje id događaja. To se radi tako da se dodaje „props“ a pod njima id te je potrebno dodati pod router-om „props: true“. Sada je potrebno kao i prije zamijeniti statične podatke iz datoteke „Event.vue“ sa dinamičkim.

4.3. Stranica za prošle događaje

Za početak izrađuje se stranica „PastEvent.vue“ koja izgleda potpuno isto kao i stranica „Events.vue“, samo na kraju datoteke se poziva getter pod nazivom „loadedPastEvents“. Za njegovu implementaciju u datoteci „index.js“ u store-u kopira se sadržaj od „loadedEvents“ te se isti postupak radi i za mutaciju „setLoadedEvents“. Sada je potrebno implementirati akciju „loadPastEvents“ koja je u početku slična kao i „loadEvents“. No razlika se nalazi unutar „for“ petlje gdje se prvo dohvaća trenutni datum te mu se dohvaćaju podaci o godini, mjesecu, danu i vremenu u drugoj varijabli zvanoj „datetime“. Varijabla „currentdatecompare“ sadrži prijašnju varijablu ali u parsiranom obliku što znači da se dobije oblik datuma koji prikazuje sekunde koje su prošle od 1.1.1970., te će se na taj način uspoređivati datumi. Iz petlje se dohvaća i datum događaja koji se na isti način parsira te se radi provjera koji je datum veći. Ukoliko je trenutni datum veći od datuma događaja, to govori kako je događaj prošao te će se takvi događaji postaviti u „events“ te na kraju pozvati mutaciju „setLoadedPastEvents“ kojoj se prosljeđuju prošli događaji. Sada je potrebno na isti način modificirati akciju „loadEvents“ tako da se kopira kod iz akcije „loadPastEvents“ te se promijeni usporedba tako da ukoliko je trenutni datum manji od datuma događaja, znači da je takav događaj nadolazeći te će se ga i prikazati. Kao završni korak potrebno je u router registrirati tu stranicu te je prikazati ukoliko je

korisnik prijavljen u datoteci „App.vue“.

```
loadPastEvents ({commit}) {
  commit('setLoading', true)
  firebase.database().ref('events').once('value')
    .then((data) => {
      const events = []
      const obj = data.val()
      for (let key in obj) {
        var currentdate = new Date()
        var datetime = currentdate.getFullYear() + '-' +
         (currentdate.getMonth() + 1) + '-' + currentdate.getDate() +
          ' ' + currentdate.getHours() + ':' + currentdate.getMinutes() +
          ':' + currentdate.getSeconds()
        var currentdatecompare = Date.parse(datetime)
        var eventdate = obj[key].date
        var eventdatecompare = Date.parse(eventdate)
        if (currentdatecompare > eventdatecompare) {
          events.push({
            id: key,
            title: obj[key].title,
```

Slika 8. Dio funkcije za usporedbu datuma

5. Implementacija aplikacije StudentEvent – forma za kreiranje događaja

U ovom poglavlju će se prikazati postupak implementacije stranice za kreiranje događaja sa formom koju korisnik popunjava kako bi kreirao novi događaj. Pri tome je potrebno paziti na datume i vrijeme te postavljanje slike za događaj.

5.1. Implementacija forme

Postupak implementacije će se provoditi u datoteci „CreateEvent.vue“ te će se ponovno koristiti dokumentacija za Vuetify pod kategorijom „text fields“. Ponovno se radi isti oblik kao i do sada sa <v-container>,<v-layout row> te <v-flex xs12 sm6 offset-sm3> komponentama. Unutar <v-flex> komponente postavlja se naslov za kreiranje sa <h4>. Nakon naslova radi se novi <v-layout row> sa <v-flex xs12> komponentom te postavlja se <form> sa novim <v-layout row> i <v-flex xs12 sm6 offset-sm3>. Unutar tih komponenti postavlja se <v-text-field> na sljedeći način:

```
<v-text-field
    name=“title“
    label=“Naslov“
    id=“title“
    required></v-text-field>
```

Name je ime komponente, id je istog naziva za identifikaciju polja, label predstavlja tekst koji će se prikazivati u aplikaciji, a required poziva već ugrađenu validaciju koja će vratiti grešku ukoliko to polje ostane prazno. Za upisivanje lokacije jednostavno se kopira dio koda za naslov te se preimenuje sve za lokaciju. Pošto još uvijek ne postoji baza podataka za upload slika, zasada će se postaviti da korisnici mogu upisati Url od slike sa interneta. Kao i do sada, kopira se dio koda za <v-text-field> i postavljaju se nazivi na „imageUrl“. Zatim se dodaje i opis događaja na isti način kao do sada, samo još je potrebno dodati „multi-line“ radi više prostora za upis teksta. Kako bi se dobio prikaz slike koju je postavljena od strane korisnika, ispod unosa slike će se

dodati još jedan red forme kao i do sada te će se postaviti <img :src="imageUrl" height> property koji trenutno ne postoji, no biti će izrađen naknadno.

Na dnu datoteke se postavlja <script> kako bi se mogao napraviti bind između inputa i property-a. Dakle, izrađuje se kao i ranije data return u kojem se samo redaju i postavljaju prazne vrijednosti za title, location, imageUrl i description. Sada je u svakom redu forme moguće koristiti v-model za spajanje sa tim vrijednostima

```
computed: {
  formIsValid () {
    return this.title !== '' &&
      this.location !== '' &&
      this.imageUrl !== '' &&
      this.description !== ''
  }
},
```

Slika 9. Provjera popunjenosti forme

varijabli. Na taj način se dohvaćaju vrijednosti koje je korisnik unio u formu. Zatim je potreban gumb za kreiranje događaja, a za to jednostavno se napravi novi red kao i prije te se postavi <v-btn> komponenta sa tekstom za kreiranje događaja. U taj gumb će se postaviti „ :disabled="!formIsValid" “ te će se ta provjera implementirati pod „computed“.

Na ovaj način gumb neće biti omogućen sve dok svi navedeni podaci nisu ispunjeni. Sljedeći korak koji je potrebno napraviti je spremiti sve te podatke u store kada se klikne gumb za kreiranje događaja. Za implementiranje te mogućnosti potrebno je raditi u datoteci index.js u mapi „store“ te unutar „mutations“ gdje se radi nova funkcija. Funkcija se naziva „createEvent“ te prima agrumente state i payload, gdje je state stanje trenutnih događaja koji su napravljeni, a payload je objekt sa podacima kojeg korisnik šalje klikom na gumb za kreiranje događaja. U toj funkciji jednostavno se sprema „payload“ na sljedeći način: state.loadedEvents.push(payload). Pod „actions“ još je potrebno odrediti koji se podaci nalaze u objektu pa je isto tako potrebna funkcija sa istim nazivom te u njoj se definira sve što je potrebno spremiti (title: payload.title itd.). Na kraju te funkcije se radi commit('createEvent', event) kako bi se poslali podaci unutar „mutations“ dijela koda. Event je taj objekt koji se šalje u drugu funkciju jer su u njemu definirani podaci koji se šalju. Sada u datoteci „CreateEvent.vue“ pod <v-btn> potrebno je dodati

„type='submit' “ te isto tako pod <v-form> se dodaje <v-form @submit.prevent=“onCreateEvent“>. Kako bi se događaj spremio kako treba, potreban je nekakav id pod createEvent u store-u. Taj id će se zasad postaviti statički pošto ne postoji id kojeg će generirati baza podataka. Isto tako će se dodati i datum koji je potrebno urediti da odgovara u formatu koji se koristi u Hrvatskoj. Zasada je implementirana aplikacija koja ima izbornike, naslovnu stranicu, popis događaja i

```
methods: {
  onCreateEvent () {
    if (!this.formIsValid) {
      return
    }
    const eventData = {
      title: this.title,
      location: this.location,
      imageUrl: this.imageUrl,
      description: this.description,
      date: new Date()
    }
    this.$store.dispatch('createEvent', eventData)
    this.$router.push('/events')
  }
}
```

Slika 10. Metode koje dodjeljuju vrijednosti objektu eventData te ga prosljeđuju funkciji 'createEvent' unutar store-a

formu za kreiranje događaja. Sljedeći je korak nadogradnja forme za kreiranje događaja koja se odnosi na datume te nakon toga će biti moguće implementirati bazu podataka.

5.2. Odabir datuma i vremena za događaj

Za odabir datuma i vremena isto tako postoje komponente u službenoj dokumentaciji [13]. Tako da iznad gumba za kreiranje novog događaja radi se novi red po istoj shemi kao i do sada, te <h4> tekst koji govori da se odaberu datum i vrijeme. Izrađuju se novi redovi kako bi se mogli postaviti <v-date-picker> i <v-time-picker format="24hr">. Na dnu datoteke je potrebno pod data return postaviti i property-e za time i date kao new Date(). Potrebno je napraviti bind, pa će se pod <v-date-picker> i <v-time-picker> postaviti v-model=“date“ i v-model=“time“. Problem se

sada javlja kod tipa podatka koji se treba poslati, odnosno način na koji se on prikazuje. U prvobitnom stanju bez ikakvih promjena datum je prikazan u string formatu prema ISO 8601 standardu što izgleda ovako: 2018-07-30T12:23:44Z. No

```
submittableDateTime () {
  const date = new Date(this.date)
  if (typeof this.time === 'string') {
    let hours = this.time.match(/^(\d+)/)[1]
    const minutes = this.time.match(/:(\d+)/)[1]
    date.setHours(hours)
    date.setMinutes(minutes)
  } else {
    date.setHours(this.time.getHours())
    date.setMinutes(this.time.getMinutes())
  }
  return date
}
```

Slika 11. Formatiranje datuma

kada se odabere neki drugi datum, format se promjeni te taj datum izgleda drugačije i tipa je date (2018-07-30). Isto vrijedi i za sate. Pod computed se postavlja funkcija submittableDateTime kako bi se regulirao i dobio željeni oblik datuma. Postavlja se novi datum koji ima vrijednost od this.date, te se može vidjeti da datum funkcionira te se prikazuje u normalnom formatu. Za sate je postupak malo kompleksniji te je potrebno u postojeći date dohvatiti sate sa setHours i setMinutes. Ukoliko je vrijeme u formatu date, mogu se odmah dohvatiti vrijednosti, no ako je string, potrebno je napraviti varijable za minute i sate. Moraju se dodijeliti vrijednosti te odrediti od kojeg znaka će se uzeti vrijednosti jer su u string formatu.

Pod eventData je potrebno za date property dodati vrijednost this.submittableDateTime kako bi se dohvatio pravi format datuma. Sada se mogu zamijeniti statični podaci unutar datoteke „Event.vue“ sa {{ event.date}} te se mogu dodati u store na statičnim događajima za datum, opis i slično.

Kod prikaza događaja koje korisnik kreira, datum je ponovno prikazan u ISO 8601 standardu što znači da će biti potreban filter za datum kako bi se prilagodio njegov prikaz. Za tu potrebu unutar „src“ mape se kreira nova mapa pod nazivom „filters“ unutar koje se radi nova datoteka „date.js“. Potrebno je napraviti export za funkciju zato što je filter samo funkcija. Ta funkcija prima vrijednost te isto tako vraća vrijednost, a to bi bili stari te novi formatirani datum. Pošto je vrijednost koja se dobije

u string obliku, potrebno je napraviti novu konstantu za datum te u nju spremiti tu vrijednost koja tada postaje datum. Vrijednost koja se vraća pomoću „.toLocaleString()“ funkcije izgleda ovako: return date.toLocaleString(['hr-HR'], {month: 'long', day: '2-digit', year: 'numeric', hour: '2-digit', minute: '2-digit'}). Pod prvim zagradama se definira koje će se lokalne informacije koristiti, a nakon nje se biraju formati za dan, mjesec, godinu, sate i slično [4]. Prelazi se u „main.js“ datoteku kako bi se napravili uvoz i registracija za filter. Prvo se dodaje „import DateFilter from './filters/date'“ te se provodi registracija sa „Vue.filter('date', DateFilter)“ naredbom. Tu je specificirano što je ulazna vrijednost i koji će se filter pozvati. Sada kada je filter registriran, postavlja ga se na sva mjesta gdje se nalaze datumi u raznim datotekama. Na primjer {{ event.date }} će postati {{ event.date | date }} .

6. Implementacija aplikacije StudentEvent – Firebase i autentifikacija

U ovom poglavlju će se prikazati postupak implementacije baze podataka Firebase koja će se instalirati i registrirati u aplikaciju te će se napraviti forma za prijavu na aplikaciju.

6.1. Postavljanje baze podataka i implementacija stranica za registraciju i prijavu

Firebase je baza podataka koju razvija Google. Nudi besplatne usluge kao što su spremanje podataka i autentifikacija. Za korištenje Firebase usluga potrebno je samo imati Google račun te vas Firebase sam vodi kroz proces kreacije baze podataka. Za vrstu autentifikacije jednostavno se odabere željeni način i stisne se gumb „enable“, a to su u ovom slučaju e-mail adresa i lozinka. Pošto je potrebno implementirati bazu, u gornjem desnom kutu se stisne gumb „Web setup“ te će se otvoriti prozor sa svim informacijama za povezivanje sa bazom podataka. Iz tog prozora se kopiraju apiKey, authDomain, databaseURL, projectID i storageBucket. Instalacija je zapravo ista kao i za Vuex, samo što se na kraju postavlja zadnja riječ „firebase“. Sada slijedi inicijalizacija baze podataka. U datoteci „Main.js“ se na početku radi uvoz na sljedeći način: „import * as firebase from 'firebase'“. Unutar new Vue radi se metoda „created“ te se u nju spremaju podaci koji su kopirani. Važno je

```
created () {
  firebase.initializeApp({
    apiKey: 'AIzaSyCrgAFSwF7xtJ0yI0kRjyXFiF6elkFFBqo',
    authDomain: 'studentevent-bd0df.firebaseapp.com',
    databaseURL: 'https://studentevent-bd0df.firebaseio.com',
    projectId: 'studentevent-bd0df',
    storageBucket: 'studentevent-bd0df.appspot.com'
  })
```

Slika 12. Podaci za inicijalizaciju baze podataka

napomenuti kako pristup i dohvaćanje podataka nije moguće samo na pregledniku Firefox zbog „IndexedDB“ standarda.

Sada kada na raspolaganju stoji baza podataka, moguće je započeti implementaciju stranice za registraciju. Za implementaciju stranice ponovno se koriste komponente <v-container>, <v-layout row> i <v-flex xs12 sm6 offset-sm3>.

Zatim se postavljaju <v-card> i <v-card-text> te unutar njih novi <v-container> koji sadrži <v-form>. Unutar forme se postavlja novi <v-layout row> i <v-flex xs12>. Prvo je potrebna nova e-mail adresa pa će se postaviti <v-text-field name="email" label="E-mail adresa" id="email" v-model="email" type="email" required>. Model na služi kako bi vrijednost koju korisnik upiše bila povezana na property kasnije, a type automatski postavlja pravila da bi unos trebao biti u obliku e-mail adrese te je dodana ugrađena validacija sa „required". Za lozinku se može kopirati taj isti oblik koji već postoji te ga se dodaje ispod e-mail adrese, samo je potrebno promijeniti informacije tako da odgovaraju za lozinku. Zatim je potrebno dodati još jedan red koji bi se trebao odnositi na to da se ponovno upiše lozinka te da ih se uspoređuje. Ispod svega se dodaje za kraj jedan <v-btn type="submit"> kako bi se mogli poslati podaci za registraciju. Na dnu datoteke ponovno se dodaje <script> te pod data () return se dodjeljuju vrijednosti koje su postavljene pod „v-model", a te će vrijednosti na početku biti prazni stringovi. Sada je potreban computed property u kojem će se nalaziti funkcija za usporedbu lozinki. Funkcija comparePasswords će imati samo jednu liniju koda, a to je da vraća „return this.password !== this.confirmPassword ? 'Lozinke nisu jednake' : ' ' ". Sada u komponenti <v-text-field> se dodaje „:rules='[comparePassowrds]'" kako bi ta funkcija radila. Zatim je potrebno postaviti metodu koja će dozvoliti submit. Zato se pod „methods" radi funkcija „onSignup ()". Zatim pod <form> se dodaje „@submit.prevent="onSignup">. Potrebna je funkcija kako bi korisnika registrirali na bazu, a funkcije se rade u „index.js" datoteci u mapi „store". No prije te funkcije, potrebno je i u trenutnoj datoteci napraviti uvoz za Firebase bazu podataka. Funkcija „signUserUp" dobiva payload od korisnika koji šalje podatke. Za funkciju se koriste Firebase ugrađene funkcije za autentifikaciju te toj istoj funkciji se šalju podaci koje je unio korisnik klikom na gumb za registraciju. Zatim bi bilo potrebno dobiti podatke koji se spremaju u Vuex store, a za to treba mutacija koja se naziva „setUser" gdje se šalju ti novi podaci korisnika. Nakon toga u funkciji „signUserUp" se postavlja konstanta „newUser" kojoj se dodjeljuje vrijednost iz Firebase baze gdje je id pod nazivom „uid" te mu se postavlja prazno polje za događaje na koje će se korisnik potencijalno prijaviti. Na kraju se dodaje „.catch()" koja služi za dohvaćanje i čitanje grešaka (error poruka). Sada se radi commit za mutaciju koja je ranije napravljena ispod polja za konstantu „newUser". Zatim se odlazi u datoteku „Signup.vue" te u funkciji „onSignup" se poziva funkcija iz store-a te

joj se prosljeđuju potrebni podaci: „this.$store.dispatch('SignUserUp, {email: this.email, password: this.password)“.

Sada je potreban getter kako bi se dohvatio trenutno prijavljeni korisnik. Prvo se ide u store i postavlja se zadani (default) user na null, tako da se ne dobiju statičke vrijednosti. Zatim se radi novi getter koji će se nazvati „user“ koji samo vraća state.user iz Vuex store-a. Prelazi se u „Signup.vue“ datoteku te se radi novi computed property koji se isto tako naziva „user“ koji zapravo radi isto kao i originalni getter, a to znači da vraća this.$store.getters.user. Nakon toga se postavlja „watch“ koji gleda svaki puta kada se promijeni stanje korisnika. Ukoliko je korisnik prijavljen, odnosno nije null, poziva se this.$router.push('/') i šalje se korisnika na početnu stranicu. Sada je potrebno da se i na izborniku mijenjaju opcije ovisno o tome je li korisnik prijavljen. Unutar datoteke „App.vue“ radi se novi computed property sa funkcijom menuItems. Prijašnje menuItems polje se miče u novu funkciju istog imena koja vraća one mogućnosti izbornika koje su predodređene za prijavljene i neprijavljene korisnike. Za to je potreban običan if i funkcija userIsAuthenticated koja vraća stanje korisnika preko ranije definiranog getter-a.

```
signUserUp ({commit}, payload) {
  commit('setLoading', true)
  commit('clearError')
  firebase.auth().createUserWithEmailAndPassword(payload.email, payload.password)
    .then(
      user => {
        const newUser = {
          id: user.uid,
          registeredEvents: [],
          fbKeys: {}
        }
        commit('setUser', newUser)
      }
    )
    .catch(
      error => {
        console.log(error)
      }
    )
},
```

Slika 13. Funkcija za spremanje korisnika na bazu podataka

Sada je potrebno urediti stranicu za prijavu koja je jako slična stranici za registraciju, pa će se kopirati cjelokupni kod iz stranice za registraciju. Briše se red za potvrdu lozinke, mijenja se tekst gumba, a metoda iz forme i u skripti se naziva „onSignIn“. Pravilo za usporedbu lozinki nije potrebno pa se briše. Sada je potrebno unutar store-a napraviti novu funkciju „signUserIn“ koja isto na slični način koristi Firebase funkciju „auth().signInWithEmailAndPassword“ te u zagradama se definira što se prosljeđuje. Funkcija u nastavku izgleda isto kao i funkcija za registraciju, hvataju se error-i i dodjeljuju se vrijednosti za korisnike za Vuex store.

6.2. Prikazivanje error poruka i znakova za učitavanje podataka

Cilj je pratiti stanje aplikacije tako da se prikazuju pripadajuće error poruke (poruke greške) i znak za učitavanje podataka u ukoliko se izvršava neka radnja. Za početak postavlja se u store-u pod „state“ loading na flase i error na null. Sada su potrebne odgovarajuće mutacije kako bi se moglo mijenjati stanje tih podataka. Radi se mutaciju „setLoading“ koja za argument dobiva trenutno stanje (state) i novo stanje koje je tipa bool (payload). Zatim se jednostavno dodjeljuje vrijednost sa „state.loading = payload“. Na isti način se radi mutacija „setError“ tako da joj se dodijeli nova vrijednost. Postaviti će se još jedna mutacija „clearError“ kojom se error-u dodjeljuje vrijednost null. Za tu funkciju treba i akcija koja poziva tu mutaciju istog imena. Sada se mogu početi koristiti novonastale funkcije. U funkciji „sigUserUp“ u prvome redu odmah se radi commit da je loading istinit ('setLoading' , true). Nakon toga je potrebno pozvati funkciju „clearError“ jer uvijek kada se šalje novi zahtjev briše se potencijalni prethodni error. Kada se dođe do bloka „.then“, loading se postavlja na false, a isto se odnosi i na „.catch“ blok. No, ukoliko je dobiven neki error, isto se radi commit kojim se sprema ta poruka u funkciju „setError“ ('setError', error). Isti proces se radi i za funkciju „signUserIn“. Sada kada se upravlja sa error porukama i procesom učitavanja podataka, potrebno im je dodijeliti komponentu <v-alert> koja se nalazi u Vuetify dokumentaciji [6].

Pošto će se ta komponenta koristiti na više mjesta, praktično je napraviti novu mapu u kojoj će se napraviti nova „Alert.vue“ datoteka. Unutar te datoteke postavlja se <template> i <v-alert error dismissable @input=“onClose“>. Zatim unutar <script> se radi metoda koja se naziva „onClose“, a ona samo emitira poruku „dismissed“

(this.$emit). Sada u „main.js“ datoteci se mora napraviti uvoz i registracija za alert na isti način kao i do sada. Naravno, kod registracije komponente mora se paziti koje će se ime uzeti, jer to tada postaje nova komponenta, a trenutni naziv je „app-alert“. Sada se može testirati kako poruke rade tako da se na početku datoteke „Signup.vue“ doda novi red sa <v-layout row> i <v-flex xs12 sm6 offset-sm3> te unutar tih komponenti se postavlja <app-alert @dismissed=“onDissmised“ :value=“true“>. Sada se treba u istoj datoteci dodati nova metoda „onDissmised“ koja poziva funkciju „clearError“ iz store-a. U store-u se zatim dodaje novi getter za error koji vraća njegovo stanje, a isto vrijedi i za loading. Sada se može dodati novi computed property u datoteci „Signup.vue“ pod nazivom „error“ jer pomoću te ćemo funkcije preko getter-a dohvaćati error. Sada kada taj property postoji, možemo ga dodati u <v-layout row v-if=“error“> gdje se nalazi <app-alert>. Unutar <app-alert> postavlja se i binding <app-alert @dismissed=“onDissmised“ :text=“error.message“>. Sada jednostavno unutar „Alert.vue“ datoteke se dodaje property pod nazivom „text“ kao bi ju povezali i prikazali za <v-alert> {{ text }} </v-alert>.

Riješila se problematika za prikazivanje error-a, još treba postaviti loading. Izvor za postavljanje loader-a se nalazi u dokumentaciji za gumbe. U datoteci

```
<v-layout row>
  <v-flex xs12>
    <v-btn type="submit" :disabled="loading" :loading="loading">
      Registracija
      <span slot="loader" class="custom-loader">
       <v-icon light>cached</v-icon>
      </span>
    </v-btn>
  </v-flex>
</v-layout>
```

Slika 14. za loading

„Signup.vue“ ispod teksta za gumb kojim se obavlja registracija, postavljamo .

Pod computed isto tako se postavlja loading koji dohvaća stanje preko getter-a. Iz iste datoteke u Vuetify dokumentaciji pod „style“ se kopira kod kako bi „custom-loader“ prikazao animaciju kako treba. Kopirati će se i stil u „main.styl“ datoteku bez <style> sintakse, već samo CSS kod. Isti postupak se radi i za stranicu prijave na aplikaciju.

7. Implementacija aplikacije StudentEvent – spremanje događaja na bazu podataka i sigurnost

Zasada aplikacija ima događaje koji, kada se kreiraju, postaju vidljivi unutar Vuex store-a. U ovom poglavlju će se razraditi postupak implementacije za spremanje događaja na bazu podataka te će se proučiti sigurnosni aspekti aplikacije.

7.1. Spremanje događaja na bazu podataka

Za ovu funkcionalnost potrebno je nadograditi funkciju „createEvent“ u store-u. Baza podataka ne sadrži tablice već je to jedan veći JSON objekt sa granama koje sadrže podatke. Prvo se iz prethodno navedene funkcije briše statični id pošto će Firebase sam generirati id. Još se treba pretvoriti datum u „payload.date.toISOString“ kako bi se spremio kao string na bazu. Zatim se sprema objekt „event“ tako da se poziva ta radnja za bazu podataka: firebase.database().ref('events').push(event). Nakon toga se radi „.then“ blok gdje se dohvaćaju ti podaci i „.catch“ blok gdje se dohvaćaju error-i. Sa „ref“ se referencira grana u koju se želi spremiti podatke, te sa „push“ naredbom se šalje objekt na bazu. Sada se postavlja problem da ne postoji id za stranicu događaja kada se klikne gumb za prikaz. Za tu potrebu mora se dohvatiti id iz baze podataka, a on već postoji pod „.then“ blokom. Kako bi se dohvatio id, radi se nova konstanta kojoj se dodjeljuje vrijednost „data.key“ koji je oblik koji podržava Firebase za id. Sada je potrebno taj key dodati u objekt koji se šalje dalje, a to se postiže sa „…event, id: key“. Na taj način se dohvaća objekt i dodaje mu se nova vrijednost. Sada kada radi funkcionalnost prikaza pojedinog događaja, moraju se dohvatiti svi događaji koji su spremljeni u bazi. Raditi će se nova akcija koja će se zvati „loadEvents“ kojom opet iz baze dohvaćamo sve događaje sa „firebase.database().ref('events').once('value')“. Ovime se poziva Firebase da ispiše popis svih podataka unutar „ref“ grane. Isto kao i do sada, dohvaćaju se error-i. Kod „.then“ bloka ponovno se dohvaća data, no sada se radi novo polje pod nazivom „events“ i radi se nova konstanta „obj“ čija je vrijednost „data.val()“ koja omogućuje pristup vrijednostima iz baze. Pošto je „obj“ objekt, on predstavlja jedan objekt u bazi podataka pod granom „events“. Kako bi se omogućilo kretanje po toj grani, mora se napraviti „for“ petlja čije se vrijednosti spremaju u polje „events“ koje je ranije izrađeno. Naravno, za pristup informaciji kroz taj loop, potrebno je koristiti „key“ kao

id, pa bi recimo naslov bio title: obj[key].title. Zatim to polje moramo proslijediti novoj mutaciji koja će se odmah i postaviti. Mutacija se zove „setLoadedEvents“ gdje polje sa događajima prosljeđujemo u state.loadedEvents.

Taj proces dohvaćanja podataka iz baze će se raditi kada se sama aplikacija učitava, a taj dio koda se nalazi u „main.js“ datoteci. Ispod inicijalizacije baze, poziva se iz store-a funkcija „loadEvents“. Zgodno bi bilo u funkciji „loadEvents“ kao i u prijašnjim funkcijama dodati „commit('setLoading, true)“ te prekinuti učitavanje kada se funkcija izvrši. Sada u „Home.vue“ je potrebno dodati loadere. Oni se isto tako nalaze u Vuetify dokumentaciji čiji će se kod kopirati te napraviti novi red prije carousel-a kako bi se kopirani kod mogao zalijepiti [14]. Za taj loader se može prilagoditi širina i veličina unutar komponente, te se dodaje „v-if='loading'“ što znači da će se samo prikazivati kada je loading postavljen na true. Za tu potrebu jednostavno se napravi novi computed property pod naziom „loading“ a on samo dohvaća stanje loading-a iz getter-a. Pod <v-layout> komponentom od carousel-a dodaje se „v-if='!loading'“ što znači da se carousel prikazuje kada je loading postavljen na false.

7.2. Sigurnost pristupa određenim stranicama

Sada kada postoji mogućnost kreiranja događaja, potreban je način da se toj stranici ne može pristupiti, osim ako je u pitanju prijavljeni korisnik. U mapi „store“ kreirati će se nova datoteka pod nazivom „auth-guard.js“ te će se u njoj odmah samo definirati funkcija sa export default koji ima argumente to, from i next što označava radnje router-a. Na vrhu datoteke radi se uvoz za store pošto se mora koristiti „if“ petlja kojom provjeravamo stanje korisnika, odnosno je li korisnik prijavljen. Ako je korisnik prijavljen, poziva se „next“ što znači da router ide dalje sa svojom namjerom, no ako korisnik nije prijavljen, isto se poziva „next“ no unutar njega se definira na koju stranicu će preusmjeriti korisnika, što je u ovom slučaju „/signin“. Sada se odlazi u datoteku „index.js“ koja se nalazi u mapi „router“ kao bi se biralo na koju će se stranicu postaviti guard. Prvo je potrebno napraviti uvoz za „AuthGuard“ kao i za sve stranice do sada. Kod stranica kojima je potrebno dodati guard, dodaje se novi property koji se naziva „beforeEnter“ te mu se dodaje vrijednost „AuthGuard“.

Sada je potrebno unutar „main.js“ datoteke provjeriti je li korisnik već prijavljen, zato što Firebase sam stvara svoj token za svaku prijavu. To se radi pomoću Firebase funkcije „firebase.auth().onAuthStateChanged“ koja vraća korisnika. Ako je korisnik prijavljen, odnosno ako nije null, poziva se funkcija 'autoSignIn' kojoj se prosljeđuju podaci za korisnika kako bi korisnika automatski prijavili na aplikaciju ukoliko postoji token. Sada se unutar store-a radi ranije navedena funkcija koja poziva funkciju „setUser“ te joj prosljeđuje id i popis registriranih događaja od trenutno prijavljenog korisnika. Unutar „App.vue“ datoteke potrebno je dodati gumb za odjavu, a to se radi sa novim gumbom <v-btn v-if=“userIsAuthenticated“ @click=“onLogout“> koji se nalazi ispod <v-btn> komponente koja sadrži „v-for“. Za izbornik kod mobilnih uređaja, radi se ispod postojećeg novi <v-list-tile> koji ima isti uvjet koji provjerava je li korisnik prijavljen te click listener. Na dnu je potrebno napraviti metodu „onLogout“ koja poziva funkciju „logout“. U store-u pod „actions“ se radi funkcija „logout“ koja samo radi commit koji postavlja korisnika na null. Isto tako potrebno je odjaviti korisnika preko Firebase funkcije kako bi se makao taj token, a to se radi sa funkcijom „firebase.auth().signOut()“. Još preostaje da dodjeljivanje property-a „CreatorId“ kako bi se znalo tko je autor događaja, a to se radi preko getter-a. Isti postupak se radi za „loadEvents“ gdje je potrebno dodati kako se izvlači i „creatorId“.

7.3. Stranica za promjenu lozinke

Za stranicu koja ima mogućnost promjene lozinke potrebno je implementirati stranicu koja će imati input polja za potvrdu trenutnih podataka i unos novih te odgovarajuću funkciju. Za dizajn stranice može se koristiti datoteka „Signup.vue“. Samo će se dodati novi redovi te preimenovati imena za komponente (oldemail, oldpassword, password, confirmpassword). Pod data return će se postaviti sve te vrijednosti na prazne stringove te će se koristiti computed property „fromIsValid“ gdje se provjerava jesu li svi podaci popunjeni te je li nova lozinka ima 6 znakova koji je minimalni zahtjev Firebase baze podataka. Idući property-i koji imaju iste funkcije kao i ranije, a to su „comparePasswords“, „error“, „loading“ i „watch“.

```
updateUser ({commit}, payload) {
  var newPassword = payload.password
  firebase.auth().signInWithEmailAndPassword(payload.oldemail, payload.oldpassword)
  .then(function (user) {
    user.updatePassword(newPassword)
    commit('setLoading', false)
    alert('Podaci su uspješno ažurirani')
  })
```

Slika 15. Akcija za ažuriranje lozinke

Pod metodama će se postaviti „onUpdateUser“ koji se nalazi unutar <form> komponente. Provjerava je li forma ispunjena do kraja te sprema podatke u konstantu te poziva „updateUser“ koji šalje tu konstantu. Akcija „updateUser“ se radi tako da se sprema nova lozinka u varijablu te se poziva Firebase kako bi se napravila prijava radi autentifikacije. Onda se unutar „.then“ bloka poziva update za novu lozinku te se onda šalje korisniku obavijest da je radnja uspješno napravljena. Isto tako, hvataju se error-i kao i u svakoj funkciji.

8. Implementacija aplikacije StudentEvent – spremanje slika na bazu podataka i mogućnosti događaja

Kod kreiranja događaja je potrebno omogućiti korisnicima da postave i prenesu vlastitu sliku sa računala, a ne samo preko linka kao do sada. Te slike se moraju spremiti na Firebase bazu podataka. U prvom dijelu poglavlja će se opisati postupak za prijenos slika, a drugom dijelu poglavlja će se postaviti gumbi i tekst za uređivanje i prikazivanje informacija o događaju.

8.1. Spremanje slika na bazu podataka

Raditi će se u datoteci „CreateEvent.vue“ gdje se briše <v-text-field> koji sadrži informacije o slici, odnosno „imageUrl“. Umjesto te komponente postavlja se <v-btn raised class=“primary“ @click=“onPickFile“> sa odgovarajućim tekstom te <v-input type=“file“ style=“display: none“ ref=“fileInput“ accept=“image/*“ @change=“onFilePicked“>. Sada će se kreirati metoda „onPickFile“ kako bi se usmjerio klik na gumb za dodavanje datoteke, dok se <v-input> ne vidi. U toj metodi se poziva klik preko imena reference koja je dodana: „this.$refs.fileInput.click()“.

```
onPickFile () {
  this.$refs.fileInput.click()
},
onFilePicked (event) {
  const files = event.target.files
  let filename = files[0].name
  if (filename.lastIndexOf('.') <= 0) {
    return alert('Odaberite validnu sliku!')
  }
  const fileReader = new FileReader()
  fileReader.addEventListener('load', () => {
    this.imageUrl = fileReader.result
  })
  fileReader.readAsDataURL(files[0])
  this.image = files[0]
}
```

Slika 16. Metode "onPickFile" i "onFilePicked"

Sada se dodaje metoda „onFilePicked“ u kojoj se koristi zadani event iz kojeg se može izvući i prikazati prenesena slika sa novom konstantom „files“ koja zapravo predstavlja polje podataka. Prvo što je poželjno jest dohvatiti ime datoteke kako bi znalo odgovara li formatu slike. Ako ime datoteke nema u svom nazivu točku, ta datoteka nije validna te se korisniku šalje poruka upozorenja. Sada je taj file potrebno pretvoriti iz binarnog u string oblik za prikaz a to će se napraviti sa JavaScript file reader-om. U tu svrhu dodaje se event listener te za „imageUrl“ se dodaje vrijednost iz rezultata file reader-a. Sada se može dodati novi property pod nazivom „image“ koji će biti na početku null. U tom property-u će biti datoteka slike koja nema promjene u sebi. Sada pošto se posjeduje vrijednost prenesene slike, potrebno promijeniti funkciju za kreiranje događaja tako da se pod „onCreateEvent“ zamijeni „imageUrl“ u „image“ jer ona sadrži datoteku slike. I može se postaviti da ako slika ne

```
let imageUrl
let key
firebase.database().ref('events').push(event)
  .then((data) => {
    key = data.key
    return key
  })
  .then(key => {
    const filename = payload.image.name
    const ext = filename.slice(filename.lastIndexOf('.'))
    return firebase.storage().ref('events/' + key + '.' + ext).put(payload.image)
  })
  .then(fileData => {
    imageUrl = fileData.metadata.downloadURLs[0]
    return firebase.database().ref('events').child(key).update({imageUrl: imageUrl})
  })
  .then(() => {
    commit('createEvent', {
      ...event,
      imageUrl: imageUrl,
      id: key
    })
  })
```

Slika 17. Modificirana funkcija za kreiranje događaja – dio za spremanje slike

postoji da se događaj ne može kreirati. Sada se prelazi u store gdje je potrebno modificirati funkciju za kreiranje događaja.

Cilj je spremiti sliku na Firebase koja će biti povezana za događaj. Prvo se briše imageUrl koji se više ne šalje na bazu, zatim se zna kako id događaja dobivamo

kroz key pa će se napraviti „return key“ te se radi novi „.then“ blok. U novom bloku se koristi key te se dohvaća ime datoteke sa „payload.image.name“ te ekstenzija sa „filename.slice(filename.lastIndexOf(.))“. Zatim se poziva „firebase.storage().ref('events'/ + key + '.' + ext).put()“ kako bi se spremila slika na Firebase storage pod imenom događaja sa ekstenzijom. Za tu radnju je potreban return kako bi se mogao napraviti još jedan „.then“ blok. Taj blok služi kako bi se napravio update na bazi za sliku, odnosno da se poveže sa prenesenom slikom. Prvo se dohvaća Url prenesene slike sa „fileData.metadata.downloadURLs[0]“ gdje je fileData varijabla koja se koristi za pozivanje ove radnje i sve se to sprema pod „imageUrl“. Za return se poziva baza kako bi se mogao napraviti update „firebase.database().ref('events').child(key).update({imageUrl: imageUrl})“. Zatim se radi još jedan „.then“ blok kako bi se napravio commit na bazu, a za to se koristi postojeći kod kojem se dodaje još „imageUrl“ koji nedostaje.

8.2. Uređivanje naslova i opisa događaja

Sada se prelazi na mogućnosti da korisnik koji je kreirao događaj može urediti naslov i opis događaja. Za tu potrebu će se kreirati nova mapa pod nazivom „Edit“ kako bi se u njemu spremili dijalozi za promjenu informacija o događaju. Zatim će se unutar te mape kreirati nova datoteka pod imenom „EditEventDetailsDialog.vue“. Zatim će se koristiti <v-dialog width=“350px“ persistent> komponenta koja predstavlja pop up prozor u Vuefity dokumentaciji [10]. Zatim unutar te komponente se dodaje <v-btn accent slot=“activator“>. Unutar tog gumba se može dodati <v-icon> za edit. Sada je taj dijalog potrebno unijeti u „main.js“ datoteku na početku sa „import EditEventDetailsDialog from './components/Event/Edit/EditEventDetailsDialog.vue'“ te ga se registrira sa „Vue.component('app-edit-event-details-dialog', EditEventDetailsDialog)“. Sada se odlazi u „Event.vue“ datoteku gdje se bira gdje će se postaviti taj dijalog, odnosno gumb za dijalog. Za prikazivanje će se postaviti ispod naslova događaja sa <template> i <v-spacer> komponentama. Ispod tih komponenti se stavlja registrirani dijalog <app-edit-event-details-dialog>. Potrebno je raditi provjeru kada će se taj gumb za uređivanje prikazati, a prikazati će se ako je korisnik autor događaja. Kako bi se to postiglo, napraviti će se novi computed property u datoteci „Event.vue“. Prvo

će se provjeriti je li korisnik prijavljen sa time da se samo vrati getter za korisnika koji nije null ili nedefiniran. Drugi property provjerava je li korisnik prijavljen, ako nije gumb se neće pojaviti, odnosno vraća se false. Ako je korisnik prijavljen, uspoređuje se id korisnika i creator id od događaja (return this.$store.getters.user.id == this.event.creatorId). Sada se može koristiti „UserIsCreator“ property pod „v-if“ za template u kojem se nalazi gumb i dijalog. Sada će se raditi na dijalogu tako da se ispod <v-btn> komponente dodaje <v-card> komponenta te unutar nje <v-layout row wrap> i <v-flex xs12>. Unutar toga se radi <v-card-title> za naslov. Zatim se radi novi red koji se odvaja sa <v-divider> te će se u njega staviti sadržaj dijaloga. Taj sadržaj ima <v-card-text> te će se kopirati <v-text-field> za title i description iz datoteke „CreateEvent.vue“ te im se daje v-model=“editedTitle“ i isto se radi za opis događaja. Nakon toga se dodaje novi <v-divider> i novi red u koji će se staviti akcije za odustajanje i potvrdu promjena. Tamo se dodaje <v-card-actions> u koji se stavljaju dva gumba <v-btn flat> koji imaju klasu za boju teksta po želji. Zatim je potrebno dobivati podatke i radili promjene na njima, a to se radi u <script> dijelu gdje se radi export default data koja ima „editedTitle“ i „editedDescription“ koji imaju vrijednosti „this.event.title“ i „this.event.description“ koje dobivamo iz props koji imaju taj „event“. Kako bi se to napravilo, potrebno je napraviti „:event='event'“ bind u datoteci „Event.vue“ gdje je <app-edit-event-details-dialog> komponenta. Sada je potrebno dodati loader koji se može kopirati iz „Home.vue“ datoteke te ga se dodaje na vrh datoteke „Event.vue“ i postavlja mu se v-if='loading'. Taj loading property se isto postavlja na dno datoteke tako da se isto kopira iz „Home.vue“ datoteke. Staviti će se i v-else unutar <v-layout> komponente kod dijaloga. Pod gumb za spremanje stavlja se „@click='onSaveChanges'“, a za zatvaranje se može napraviti novi data property pod nazivom „editDialog“ te ga se postavlja na false unutar <v-dialog v-model=“editDialog“>. Na gumb za odustajanje se postavlja „@click='editDialog = false'“.

Sada kada postoji gumb za zatvaranje, mora se dovršiti gumb za spremanje promjena čiji je postupak teži. Radi se metoda „onSaveChanges“ gdje se isto editDialog postavlja na false te se postavlja prije toga jedan if gdje se provjerava jesu li polja za naslov i opis prazna. Ako jesu, izlazi se iz funkcije to jest ne spremaju se promjene. Za daljnji razvoj unutar store-a treba nova funkcija pod nazivom „updateEventData“ gdje se loading postavlja na true. Napraviti će se prazni objekt

pod nazivom „updateObj“ te se provjerava postoje li kod podataka koje dobivamo neke promjene. Ako promjene postoje, dodjeljuje se vrijednost tom objektu, a vrijednost je iz payload-a. Nakon toga se poziva Firebase i kao do sada te se radi update tako da se šalje „updateObj“. Nakon toga se radi „.then“ blok gdje se loading stavlja na „false“ te se radi commit na mutaciju „updateEvent“. Mutacija „updateEvent“ sadrži konstantu koja traži odgovarajući događaj pomoću find gdje uspoređuje događaje iz state-a sa id-om događaja iz payload-a te mu dodjeljuje vrijednosti iz payload-a ukoliko on postoji. Još na kraju kao i uvijek se hvataju error-i. Sada se pod „onSaveChanges“ poziva tu funkcija za update te joj se prosljeđuju potrebni podaci za promjenu.

8.3. Uređivanje datuma i vremena događaja

Pristup za implementaciju je sličan kao i u prethodnom slučaju. Napraviti će se novu datoteka pod nazivom „EditEventDateDialog.vue“ te će se kopirati sadržaj iz datoteke za uređivanje detalja o događaju. Briše se <v-icon> jer je potrebno da to bude običan gumb i mijenja se naslov po potrebi. Unutar te komponente je potreban date picker. Brišu se nepotrebni redovi tako da samo ostane jedan prazan red u koji se postavlja <v-date-picker v-model=“editableDate“ style=“width: 100%“ actions>. Unutar te komponente se postavlja <template scope=“{save, cancel}“> te dvije <v-btn> komponente. Prvi gumb je <v-btn class=“blue—text darken-1“ flat @click.native=“editDialog = false“>, a drugi gumb ima isti izgled osim što se nakon klika poziva „onSaveChanges“. Na dnu se ponovno radi <script> gdje se dodaje „event“ pod props i pod data return „editDialog“ koji ima vrijednost „false“ i „editableDate koji ima vrijednost „null“. Zatim se radi metoda „created“ koja dodjeljuje vrijednost datuma da bude onaj datum koji odgovara događaju. Metoda „onSaveChanges“ ima konstantu „newDate“ u koju se sprema trenutni datum događaja i „newDay“ koji dohvaća novoodabrani dan događaja. Isti proces se radi za mjesec i godinu. Zatim se radi update za „newDate“ tako da mu se dodjeljuju vrijednosti za dan, mjesec i godinu. Još je potrebno pozvati funkciju „updateEventData“ kojoj se prosljeđuje novi datum i id događaja. Kao i prijašnji dijalog, potrebno ga je unijeti i registrirati u „main.js“ datoteci. Sada da bi taj gumb bio

vidljiv, tu će se nakon datuma unijeti dijalog kojemu će se napraviti bind „:event=“event“ te provjera „v-if='userIsCreator'“.

Za dijalog za promjenu vremena, prvo će se napraviti nova datoteka te će se kopirati sav sadržaj datoteke za uređivanje datuma. U toj datoteci se mijenja tekst te se postavlja <v-time-picker> komponenta. Jedine veće promjene su u metodi „onSaveChanges“ gdje se sada dohvaćaju sati i minute, a kod za dohvaćanje se može kopirati iz „CreateEvent.vue“ datoteke. Još je preostalo unijeti i registrirati taj dijalog i unutar „Event.vue“ datoteke dodati taj dijalog na isti način kao i prijašnji.

8.4. Trenutno prijavljeni korisnik

Za prikazivanje trenutno prijavljenog korisnika unutar alatne trake, potrebno je napraviti getter unutar store-a koji dohvaća e-mail adresu. Koriste se dvije varijable, a to su „user“ koja sadrži podatke trenutnog korisnika iz baze podataka i „email“ koja dohvaća e-mail adresu od tog korisnika. Nakon toga se unutar datoteke „App.vue“ radi computed property koji će preko getter-a pozivati i vratiti e-mail adresu korisnika. Još je potrebno pozvati i prikazati taj e-mail unutar alatne trake tako da se vidi u cijeloj aplikaciji. Taj podatak se poziva i postavlja prije izlistavanja ostalih stavki putem „v-for“ petlje. Potrebno je postaviti da se prikazuje samo ako je korisnik prijavljen (v-if=“UserIsAuthenticated“) te da vodi na stranicu za profil (router to="/profile").

8.5. Kapacitet, autor događaja i broj prijavljenih korisnika

Slijedeća informacija koja je potrebna za kreiranje događaja je kapacitet. Kod konstante „eventData“ dodaje se „capacity“ na isti način kao i ostali podaci te unutar forme se dodaje jedan <v-layout row> za kapacitet te unutar <v-text-field> se dodaje „type='number'“ kako bi taj dio forme samo prihvaćao brojeve. Kao zadnji dio provjere, dodaje se još jedan red koji će se samo prikazivati ako je „v-if='formIsValid'“, odnosno ako je cijela forma popunjena.

Za upravljanje i odlučivanje koji su datumi omogućeni za odabir, pod „mounted“ rade se konstante koje sadrže današnje datume te im se ovisno o potrebama dodaju ili oduzimaju dani, a u ovom slučaju je omogućeno da je minimalni

dan sutrašnji (date.getDate() +1) te maksimalni jednu godinu unaprijed (date.getDate() +365). Te vrijednosti se stavljaju kao min i max za „lastFiveDays“ koji je postavljen pod „export default data“ na null. To će se implementirati tako se dodaje <v-date-picker :allowedDates = lastFiveDays>.

Sada je potrebna mogućnost da se prikaže tko je autor događaja, a to se dohvaća unutar funkcije „createEvent“ u store-u. Preko Firebase baze se dohvaća trenutni korisnik (firebase.auth().currentUser) te se ti podaci spremaju u konstantu „user“. Radi se nova konstanta „email“ kojoj se dodaje vrijednost „user.email“. Sada se ta konstanta samo dodaje u konstantu „event“ za spremanje na bazu podataka. Zatim se u tu konstantu dodaje i „signedUpUsers“ koji predstavlja broj prijavljenih korisnika na događaj, a taj broj se odmah na početku postavlja na 0. Za prikazivanje kapaciteta, autora događaja i broj prijavljenih korisnika u datoteci „Event.vue“, samo se dodaju „{{ event.capacity }}“, „{{ event.creatorEmail }}“, „{{ event.signedUpUsers }}“ u redak po želji. Za uređivanje kapaciteta u datoteci „EditEventDetailsDialog.vue“ dodaje se redak za uređivanje kapaciteta te se dodaje „editedCapacity“ pod data return i pod metode, što je zapravo isti postupak kao i kod prijašnjih dijelova dijaloga.

Dodavanje i brisanje prijavljenih korisnika se odvija kod prijave i odjave sa

```
addAPerson ({commit}, payload) {
  commit('setLoading', true)
  const updateObj = {}
  updateObj.signedUpUsers = payload.signedUpUsers + 1
  firebase.database().ref('events').child(payload.id).update(updateObj)
    .then(() => {
      this.$store.dispatch('loadPastEvents')
      this.$store.dispatch('loadEvents')
      this.$store.dispatch('loadAllEvents')
      commit('setLoading', false)
    })
    .catch(error => {
      console.log(error)
      commit('setLoading', false)
    })
},
```

Slika 18. Funkcija za dodavanje prijavljenih korisnika na događaj

događaja, odnosno pod tim gumbima. U datoteci „RegisterDialog.vue“ se samo kod pozivanja funkcija, ovisno o tome je li se korisnik prijavio ili odjavio, poziva funkcija „addAPerson“ ili „removeAPerson“. Kod funkcije „addAPerson“ se postavlja loading

na „true“ te se postavlja konstanta „updateObj.signedUpUsers = payload.signedUpUsers + 1“. Zatim se preko baze podataka radi update za taj događaj gdje se povećava broj prijavljenih korisnika. Poziva se učitavanje svih događaja, te se postavlja loading na false i hvataju se error-i. Funkcija „removeAPerson“ je potpuno ista, samo se oduzima jedno mjesto za prijavu na događaj.

8.6. Brisanje događaja

Brisanje događaja je još jedan gumb kojim se omogućuje korisnicima da izbrišu svoj događaj ukoliko su autori istog. Za tu potrebu se radi novi dijalog pod nazivom „DeleteEventDialog.vue“, a kod će se kopirati iz dijaloga „RegisterDialog.vue“. Tekst dijaloga se prilagođava prema potrebama te se pod „onAgree“ poziva funkcija „DeleteEvent“ kojoj se prosljeđuje id od trenutno odabranog događaja. Funkcija isto kao i svaka postavlja loading i hvata error-e na isti način. Nakon što se funkcija izvrši, poziva se učitavanje događaja te se preusmjerava korisnika na naslovnu stranicu. Preko baze podataka pronalazi se događaj putem „fbKey“ konstante koja sadrži id događaja te se taj događaj briše.

```
DeleteEvent ({commit}, payload) {
  commit('setLoading', true)
  const fbKey = payload
  firebase.database().ref('/events/').child(fbKey)
    .remove()
    .then(() => {
      commit('setLoading', false)
      location.reload()
    })
    .catch(error => {
      console.log(error)
      commit('setLoading', false)
    })
},
```

Slika 19. Funkcija za brisanje događaja

9. Implementacija aplikacije StudentEvent – prijava i odjava događaja i dohvaćanje stanja korisnika

U ovom poglavlju će se prikazati postupak implementacije funkcionalnosti za prijavu i odjavu na događaj kako bi se bilježili događaji za koje su se korisnici prijavili ili odjavili. Taj gumb će se prikazivati samo ukoliko je korisnik prijavljen te ako isti nije autor događaja.

9.1. Prijava i odjava događaja

Za tu potrebu će se napraviti nova mapa sa datotekom „RegisterDialog.vue“. Isto tako se može kopirati kod iz prijašnjih dijaloga te ih prilagoditi prema potrebama. Iz te datoteke se tada briše kod unutar <script> te se na vrhu unutar <v-dialog> briše „width“ i postavlja se „v-model='registerDialog'“. Pod <script> se dodaje computed property pod nazivom „userIsRegistered“ kako bi se provjerilo je li korisnik registriran kako bi se prikazao gumb za prijavu ili odjavu. Preko getter-a se vraćaju registrirani događaji od korisnika te ih se pretražuje sa JavaScript funkcijom „findIndex“ za eventId kojeg će se dodati pod „props“. Ako rezultat pretraživanja bude veći od 0, pronađen je odgovarajući id događaja.

```
computed: {
  userIsRegistered () {
    return this.$store.getters.user.registeredEvents.findIndex(eventId => {
      return eventId === this.eventId
    }) >= 0
  }
},
```

Slika 20. Pretraživanje za registrirane događaje

Pod naslovom za dijalog, postavlja se {{ userIsRegistered ? 'Unregister' : 'Register' }}. Na taj način se prikazuje gumb ovisno o tome je li korisnik registriran. U tekstu za dijalog isto se prikazuje tekst ovisno o tome je li korisnik registriran. Za te komponente se rade jednostavne provjere <v-card-title v-if=“userIsRegistered“> te za još jednu komponentu <v-card-title v-else>. Zatim će se postaviti <v-divider> i <v-card-text> gdje će se napisati tekst po želji. Sada su potrebne dvije <v-card-actions> komponente za potvrdu ili odustajanje od radnje. Unutar te komponente postavljaju

se dvije <v-btn> komponente za odustajanje (class=“red—text darken-1“ flat @click=“registerDialog = false“) i za potvrdu (class=“green—text darken-1“ flat @click=“onAgree“). Kao i za prijašnje dijaloge, isti je potrebno registrirati kao globalnu komponentu u „main.js“ datoteci pod nazivom „app-event-register-dialog“. Isti taj dijalog se mora postaviti u datoteku „Event.vue“ tamo gdje je <v-btn> komponenta za registraciju uz bind „:eventId='event.id'“ i provjeru je li korisnik prijavljen ili ako nije autor događaja. Još se mora postaviti data return koji taj dijalog postavlja na „false“. Na dnu datoteke se mora postaviti i metoda „onAgree“ kako bi se izbjegao error.

Sada je potrebna nova funkcija unutar store-a pod nazivom „registerUserForEvent“. Prvo se postavlja loading na „true“ te se onda šalje zahtjev na bazu podataka za promjenu polja za registrirane događaje. Preko baze se pristupa registracijama korisnicima, te se radi „push“ za payload koji predstavlja id događaja. Onda se radi „.then“ blok koji će postaviti loading na false te pozvati novu mutaciju istog imena kojoj će se proslijediti „fbKey“ koji ima vrijednost „data.key“, te „id“ koji ima vrijednost „payload“. Isto tako se postavlja „.catch“ za hvatanje error-a. Sada će se napraviti ranije spomenuta mutacija. Prvo se mora provjeriti je li korisnik registriran na događaj, a to se radi sa „findIndex“ funkcijom te ako je vrijednost veća od nula, izlazi se iz mutacije. Zatim se u „state“ radi „push“ za id te se isto radi i za „fbKeys“ i njegov id iz payload-a. Zatim se može odmah i napraviti mutacija iz odjavu sa događaja, a u konstantu će se spremiti registrirani događaji korisnika, te će se napraviti „splice“ za jedan događaj za koji se ponovno radi „findIndex“. Nakon toga je potrebno koristiti JavaScript „Reflect“ API koji sadrži „deletey“ kojem se prosljeđuje „fbKeys“ i payload. Sada se mora napraviti funkcija za odjavu sa događaja tako da se postavi loading na „true“ te se radi konstanta „user“ u koju se sprema stanje korisnika iz getter-a. Zatim se provjerava ima li korisnik „fbKeys“ te ako nema, izlazi se iz funkcije. Zatim se dohvaća taj „fbKeys“ te se poziva Firebase kako bi se došlo do te registracije „firebase.database().ref('/users/' + user.id + '/registrations').child(fbKey).remove “. Još se radi „.then“ blok u kojem se postavlja „loading“ na false i poziva se mutacija istog imena kojoj se prosljeđuje payload. I ponovno se radi „.catch“ za error-e.

Sada se prelazi u „RegisterDialog.vue“ datoteku gdje će se u metodi „onAgree“ raditi provjera je li korisnik registriran. Ako je, poziva se akcija za odjavu kojoj se prosljeđuje id događaja. Pod „else“ se poziva akciju za prijavu (registraciju) na događaj. Zatim se postavljaju prazni „fbKeys“ u funkcije „signUserUp“, „autoSignIn“ i „signUserIn“. Sada se postavlja problem što učiniti kada se napravi refresh na aplikaciji, a događaji za koje se korisnik registrirao se ponovno učitavaju na novo jer funkcije učitavaju prazna polja za korisnike.

```
registerUserForEvent ({commit, getters}, payload) {
  commit('setLoading', true)
  const user = getters.user
  firebase.database().ref('/users/' + user.id).child('/registrations/')
    .push(payload)
    .then(data => {
      commit('setLoading', false)
      commit('registerUserForEvent', {id: payload, fbKey: data.key})
    })
    .catch(error => {
      console.log(error)
      commit('setLoading', false)
    })
},
```

Slika 21. Funkcija za prijavu na događaj

```
unregisterUserFromEvent ({commit, getters}, payload) {
  commit('setLoading', true)
  const user = getters.user
  if (!user.fbKeys) {
    return
  }
  const fbKey = user.fbKeys[payload]
  firebase.database().ref('/users/' + user.id + '/registrations/').child(fbKey)
    .remove()
    .then(() => {
      commit('setLoading', false)
      commit('unregisterUserFromEvent', payload)
    })
    .catch(error => {
      console.log(error)
      commit('setLoading', false)
    })
},
```

Slika 22. Funkcija za odjavu sa događaja

9.2. Dohvaćanje registriranih događaja za korisnika

U „index.js“ datoteci u store-u se treba postaviti još jednu akciju koja je povezana sa „autoSignIn“ akcijom. Ta akcija se naziva „fetchUserData“ te kao i uvijek postavlja se loading na „true“ te se poziva Firebase kako bi se dohvatile sve registracije za jednog korisnika samo jednom, zato se na kraju dodaje „once“. Onda se radi „.then“ blok u kojem sa konstantom „dataPairs“ dohvaćamo i dodjeljujemo vrijednost tih podataka. Postaviti će se „registeredEvents“ kao prazno polje te preko „for“ petlje će se raditi „push“ na „key“ od „dataPairs“. Isto tako se radi prazni objekt „swappedPairs“ pošto je potrebno okrenuti redoslijed parova podataka, te se tom objektu dodjeljuje vrijednost „key“. Zatim se radi nova konstanta „updatedUser“ te se tom objektu dodjeljuju vrijednosti za id, registrirane događaje i fbKeys. Nakon toga se samo postavlja loading na „false“ i poziva se „setUser“ kojem se šalje „updatedUser“. I na kraju se radi „.catch“ za error-e. Ta se funkcija poziva kod pokretanja aplikacije u „main.js“ datoteci.

```
fetchUserData ({commit, getters}) {
  commit('setLoading', true)
  firebase.database().ref('/users/' + getters.user.id + '/registrations/').once('value')
    .then(data => {
      const dataPairs = data.val()
      let registeredEvents = []
      let swappedPairs = {}
      for (let key in dataPairs) {
        registeredEvents.push(dataPairs[key])
        swappedPairs[dataPairs[key]] = key
      }
      const updatedUser = {
        id: getters.user.id,
        registeredEvents: registeredEvents,
        fbKeys: swappedPairs
      }
      commit('setLoading', false)
      commit('setUser', updatedUser)
    })
```

Slika 23. Funkcija za dohvaćanje informacija o korisniku

Kao završna radnja radi preglednosti i bolje organizacije programskog koda, poželjno je razdvojiti store na state, mutacije, akcije i getter-e koji odgovaraju događajima i korisniku. Ukoliko postoje zajednički dijelovi koda, poželjno je i njih odvojiti u zasebnu datoteku.

10. Zaključak

U ovome radu je prikazano kako postoji više sučelja i načina pomoću kojih se može implementirati aplikacija. Isto tako se pojasnilo kako aplikacije imaju vrlo velik i rasprostranjen utjecaj na današnje društvo te predstavljaju jedan od najkorištenijih i najunosnijih oblika poslovanja. Kvaliteta i brzina aplikacije utječu na percepciju korisnika prema aplikacija te su presudni faktor za popularnost i upotrebljivost aplikacije.

Sučelje Vue.js predstavlja jedan od novijih sučelja za razvoj na bazi programskih jezika HTML i JavaScript. Upravo je programiranje preko sučelja danas najpopularniji način za razvoj web aplikacija te je za svakog developera preporučljivo koristiti prilagođena sučelja za razvoj. U aplikaciji su osim Vue.js sučelja korišteni Vuex store management, Vuetify te Firebase baza podataka kao besplatni izvor za pohranjivanje podataka koje ne ovisi o lokalnom poslužitelju. Vuetify je korišten za komponente i gradnju izgleda stranice u kombinaciji sa HTML jezikom. Vuex store management se koristio kako bi se lokalno spremali podaci, a Firebase je služio kao baza za dohvaćanje podataka.

Za usmjeravanje i registraciju puta za stranice, korišten je router koji je ugrađen u Vue.js. Funkcije su pisane u JavaScript datotekama te su povezane sa Vue datotekama sa <script> te se na taj način prosljeđuju podaci između te dvije vrste datoteka. Takav način rada je omogućio da se jednostavno upravlja podacima i da se pišu funkcije prema potrebi. State služi za prikaz kako bi objekt za događaje trebao izgledati, odnosno koje podatke sadrži. Akcije rade promjene nad podacima, a mutacije primjenjuju te promjene. Getter-i služe za dohvaćanje stanja korisnika i njima sličnih podataka. Zaključuje se kako je Vue.js sučelje uz nekoliko dodatnih paketa i proširenja moćni alat koji omogućuje brzo i efektivno razvijanje i testiranje aplikacija za različite primjene. Vue.js sučelje kao i njegovi ostali paketi i proširenja su i dalje pod aktivnim razvojem te isti često dobiva ažuriranja i nadogradnje koje donose nove mogućnosti ili proširuju korištenje postojećih mogućnosti.

Najveći nedostatak kod procesa razvoja cijele aplikacije jest nemogućnost podržavanja preglednika Firefox za spajanje na bazu podataka Firebase. Sučelje Vue.js isto tako ima manji razvojni tim od konkurencije pa je razvoj samog sučelja

usporeniji. Prednost aplikacije je što baza podataka nije ovisna o lokalnom serveru te je dostupna na bilo kojem uređaju koji ima pristup na internet. Korištenje aplikacije je zamišljeno na razini jednog sveučilišta s mogućnošću da se ta razina poveća na državnu gdje bi se aplikacija koristila od strane svih studenata. Trenutno ograničenje aplikacije je što nedostaje više mogućnosti za uređivanje profila. Korisnici se mogu samo identificirati prema e-mail adresi, no bilo bi poželjno da korisnici mogu postaviti vlastitu sliku sa nekim osobnim podacima i preferencijama. Spominju se preferencije jer bi kao dobra ideja bila i kategorizacija događaja tako da korisnicima koji preferiraju određene kategorije mogu dolaziti e-mail notifikacije o novim događajima. Za daljnji razvoj bi funkcionalnost komentiranja i ocjenjivanja događaja isto bila dobro proširenje koje bi upotpunilo korisnički doživljaj aplikacije.

Aplikacija je postavljena na GitHub hosting te je dostupna na adresi https://imatak.github.io/. Registracija i prijava na aplikaciju se može napraviti sa e-mail adresom i lozinkom po želji te aplikacija već sadrži testne korisnike i događaje.

11. Literatura

[1] Filipova, O.: „Learning Vue.js 2“, Packt Publishing Ltd., Birmingham, 2016.

[2] Kaluža, M., Troskot, K., Vukelić, B: „Comparison of Front-End Frameworks for Web Applications“, Zbornik Veleučilišta u Rijeci, Vol. 6, 2018.

[3] Material.io, (n.d.), Icons, https://material.io/tools/icons/?style=baseline, 31.7.2018.

[4] MDN web docs, (n.d.), Date.prototype.toLocaleString(), https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/toLocaleString, 10.8.2018.

[5] Peters, K: „*Vue.js review of 2017*“, 2017. URL: https://medium.com/@kevin_peters/what-happened-to-vue-js-in-2017-aeaaa69c2c6f

[6] Vuetify, (n.d.), Alert, https://vuetifyjs.com/en/components/alerts, 12.8.2018.

[7] Vuetify, (n.d.), Button, https://vuetifyjs.com/en/components/buttons, 28.7.2018.

[8] Vuetify, (n.d.), Card, https://vuetifyjs.com/en/components/cards, 4.8.2018.

[9] Vuetify, (n.d.), Carousel, https://vuetifyjs.com/en/components/carousels, 4.8.2018.

[10] Vuetify, (n.d.), Dialog, https://vuetifyjs.com/en/components/dialogs, 12.8.2018.

[11] Vuetify, (n.d.), Grid System, https://vuetifyjs.com/en/layout/grid, 4.8.2018.

[12] Vuetify, (n.d.), List, https://vuetifyjs.com/en/components/lists, 1.8.2018.

[13] Vuetify, (n.d.), Pickers, https://vuetifyjs.com/releases/0.15/#/components/pickers , 9.8.2018.

[14] Vuetify, (n.d.), Progress, https://vuetifyjs.com/en/components/progress, 12.8.2018.

[15] Vuetify, (n.d.), Quick Start, https://vuetifyjs.com/en/getting-started/quick-

start, 25.7.2018.

[16] Vuetify, (n.d.), Theme, https://vuetifyjs.com/en/style/theme, 1.8.2018.

[17] Vuetify, (n.d.), Toolbar, https://vuetifyjs.com/en/components/toolbars, 27.7.2018.

[18] Vuex, (n.d.), What is Vuex?, https://vuex.vuejs.org/, 8.8.2018.

[19] Vue.js, (n.d.), Installation, https://vuejs.org/v2/guide/installation.html, 25.7.2018.

12. Popis slika

13. Sažetak

Vue.js je sučelje koje omogućuje programiranje i razvoj aplikacija kombinacijom HTML i JavaScript jezika. Aplikacija StudentEvent osim sučelja Vue.js koristi paket za dizajn komponenti Vuetify, Vuex store management i bazu podataka Firebase. U uvodnom dijelu se govori općenito o mobilnim i web aplikacijama te se istražuje tržište i motivacija za implementaciju aplikacije. Kroz slijedeća se poglavlja prati razvoj i implementacija počevši od osnovnog oblika aplikacije. Zatim se izrađuju stranice na kojima se postepeno implementiraju mogućnosti te se podaci povezuju i prikazuju preko baze podataka. Aplikacija služi kao primjer razvojnih mogućnosti okruženja Vue.js i njegovih paketa.

Ključne riječi: Vue.js, Vuetify, Firebase, Baza podataka, Vuex, Korisničko sučelje, Implementacija komponenti, JavaScript, HTML

14. Abstract

Vue.js is an interface which provides the ability for application programming and development using the combination of HTML and JavaScript programming languages. Besides using the Vue.js interface, the StudentEvent application also uses the Vuetify component design package, Vuex store management and the Firebase database. The introductory chapter focuses on mobile and web applications in general and it also explains the general idea and motivation for developing such an application. The next chapters follow the development and implementation of the application starting with the basic design. The majority of chapters then focus on creating and connecting the pages and their data with the database. The application serves as an example of the development abilities of Vue.js and its packages.

Keywords: Vue.js, Vuetify, Firebase, Database, Vuex, User interface, Component implementation, JavaScript, HTML